A User Study of an Educational Video System

by

Caitlin R. Johnson

S.B., Computer Science and Engineering. M.I.T., 2009

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 20, 2011
[June 2011]

Author [illegible]
Department of Electrical Engineering and Computer Science
May 20, 2011

Certified by [illegible]
Dr. Christopher J. Terman
Thesis Supervisor

Accepted by [illegible]
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

A User Study of an Educational Video System

by

Caitlin R. Johnson

S.B., Computer Science and Engineering. M.I.T., 2011

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology
May 20, 2011

ABSTRACT

This thesis describes the creation of an educational video system and the results of introducing it in a large MIT class. Experience shows that there is a high demand for recorded, course-specific, educational content. While there are several solutions for recording and sharing general instructional interactions, there are not as many are not many easy ways for instructors to record and share individual interactions. The system is meant to supplement existing course material with recordings of these interactions.

Thesis Supervisor: Christopher J. Terman

Title: Senior Lecturer, Department of Electrical Engineering and Computer Science

Acknowledgements

I would like to thank Chris Terman for being an excellent academic advisor and for helping me get through both degrees at MIT. I also would like to thank him for providing just as much guidance and support as my thesis supervisor. Despite various setbacks, Chris has always had the patience and wisdom to help me stay on track, and for that I am extremely grateful.

For giving me an opportunity to teach, for keeping me around despite my various imperfections for all nine semesters, and for sharing his seemingly infinite wisdom about teaching, wine cork topology, finite state machines, and everything else, I would like to thank Steve Ward. Steve allowed me to use his course, 6.004, as a testing environment for my thesis project, and I am very thankful for that opportunity as well.

For her infallible assistance with navigating requirements, and for providing all kinds of advice on what was frequently short notice, I would like to thank Anne Hunter.

For being the first ones to inspire me to teach at MIT, for being fantastic summer office mates, and for helping me out with my research project when I got stuck, I would like to extend my sincere appreciation to Hubert Pham and Justin Mazzola Paluska.

Last, but certainly not least, I would like to thank all of the TAs and LAs who humored me and tried recording videos to add to the course content and help me with my thesis. I would also like to thank all of the students who tried out the system. In particular, I sincerely appreciate the efforts of the early adopters who provided me with useful feedback, and demonstrated admirable good humor while I worked out bugs before introducing the system to the entire class.

Table of Contents

List of Figures:

Chapter 1: Introduction

The primary goal of this project was to provide a way to record interactions between students and teachers that would otherwise be forgotten. This creates a sense of being "present" in a course that students might not otherwise have with traditional text-based methods that are available to instructors for answering questions because the videos can provide more meaningful explanations. It also allows students to see the kinds of explanations that occur between instructors and other students, which can help students to feel more comfortable with asking their own questions, in turn.

While other work had focused on capturing the experience of being in a classroom, I chose to focus on capturing one-on-one office-hours type interactions between students and teachers. In my experience helping students in one particular course for nine semesters, I have found that many students have similar questions every semester. Teachers invest a significant amount of time and energy in developing their abilities within a given subject, and it seemed a shame that this expertise in teaching often gets lost after a teacher leaves at the end of one or more semesters. Some of the most valuable teaching moments occur somewhat spontaneously when students ask for help or clarification, and only those students have the opportunity to remember these interactions afterwards. Preserving these interactions could significantly add to the educational experience of all students in the classroom, and it also might help new teachers learn effective ways to communicate information to students.

In order to record these student-teacher interactions, I selected a set of recording technologies and created a web-based content distribution system. I recorded material, and invited other instructors to contribute additional material. I granted students to this system for a semester in one particular course that I was involved in teaching. This thesis discusses the design choices behind the recording setup, the content delivery system architecture, and the results or this user study in an MIT classroom.

Chapter 2: Previous Work

There has been a good deal of previous work in publishing educational content, and many education programs offer resources in the form of lecture videos or online forums. Much of this work has focused on improving distance education – either in closed enrollment settings or publicly available self-paced material. In this section, I provide more information about examples of current strategies. First, I discuss MIT's publicly available course archives, which frequently include lecture videos. Second, I cover a more localized lecture video recording project where media was made immediately available to students in the class. Third, I describe a different approach offered by the Khan Academy, where the publicly available videos are general tutorials rather than formal lectures.

2.1 OCW

MIT's Open CourseWare initiative, commonly referred to as "OCW," allows people from around the world to access assignments and lecture material from many MIT classes. Every class with an OCW record at MIT is developed individually, and there is a resulting disparity in the level of educational content provided for every course. Class records on OCW are relevant to a particular semester. This means that for a single semester, one course might have all lectures recorded in a web-enabled video format, and these lectures would be posted as part of the class record. Homework assignments, exams, and lecture notes from that same semester are often posted, but the exact pattern is not always consistent.

OCW is a significant undertaking that benefits many people around the world. It also requires significant funding. The OCW site [1] mentions that each course requires between $10,000 and $15,000 dollars per semester to turn that term into an OCW archive, and including video content can double this approximation. This funding helps to make OCW a well-organized resource. It also supports translation for course materials, and many class records are available in multiple languages.

There are several primary differences between OCW and the system described in this thesis. Notably, OCW provides a central repository for course resources and does not assume a separate course website. Each OCW record is meant as an archive of a course website, and contains generic lecture and homework records. The records do not contain records of personal interactions between students and instructors. The OCW records are not media based, but rather centered on providing a simple set of links to archived material. In addition, each OCW archive is statically linked to a course as it was taught in a particular semester. The system that I developed and describe in this thesis provides support for organizing media for a specific course across multiple semesters. In this way, a student becomes more situated in a subject as it has been taught for several semesters, because records of instructor explanations are all accessible in a single place.

2.2 6.004 Lecture Recordings

"6.004" was the course were I conducted the user study (see Section 5.1), and it was particularly interesting to consider lecture videos made for the same course as another resource in this area. For more than one previous semester, the lecturer used a separate camera and screen recording software to capture two movies – one of the lecturer and chalkboards, and the other of the PowerPoint slides and pointer movement for calling attention to parts of the slides. This setup also involved a separate wireless microphone. These videos required some post processing to join together, and they were usually synchronized by manually finding the right start and end times for the identical audio track that they both shared. Because the lectures were mostly PowerPoint based, this more imprecise approach seemed to work well because the PowerPoint slides didn't change as rapidly as a display might if it was showing an instructor writing on a tablet. This approach also differed from the one presented in this thesis in that for more slow-moving PowerPoint slides, the lecturer found the additional video track of a human being moving around and gesturing to be more helpful.

In the tutorial videos developed in the course of this thesis, the more dynamic content was presented in the single drawing surface, and merging multiple video recordings was not as necessary for this reason, and in fact, more movement in another section of the video file would have probably been distracting. These videos were similar to those produced for this thesis in that they were made immediately available to students taking the class that semester, although they were not necessarily intended as supplements to a publicly available resource like MIT's OCW. I did reuse these videos within the class tutorial system and found them to be popular and helpful to students in a different and complimentary way.

2.3 Khan Academy

The Khan Academy is not-for-profit educational organization that is responsible, primarily, for the creation of several thousand educational tutorial videos on a publicly accessible website. Subsequently, the Khan Academy had come forth with interactive exercises, measures to track student progress through such exercises, and more advanced student-monitoring tools. Salman Khan is the creator of the Khan Academy, and he is the primary author of these tutorials.

The video explanations on the Kahn Academy website are similar in style to those created for this project. The explanations are focused on the kinds of drawings an instructor might make at a chalkboard or on a piece of paper while interacting with an individual student.

This is very similar to the approach used in the tutorial video system that I created for this project, but the most striking difference is that I tried to create a more localized solution. The tool that I developed could be used in any classroom, but it is meant as more of a record of in-person explanations, whereas Mr. Kahn has suggested that his lecture videos be used in the place of traditional lectures. Both systems are not mutually exclusive in any way, and they would almost certainly complement one another very well. The Kahn Academy videos are licensed under a Creative Commons License that would allow educational re-use in a system like the one that I have created, but even more seamless integration with the rest of the Kahn Academy tools may be possible in the future.

I developed the idea for the tutorial video system before discovering the Kahn Academy. This means that at least two people thought this was a good approach to publishing online educational media, and it also means that I had a different approach. Not only did I have a different vision for the types of recordings that would be made, but I also ended up with a different set of recording technologies that helped to achieve this vision. Mr. Kahn uses Camtasia recording software, free tablet input software, and a smaller Wacom Bamboo tablet [2] . For this project, I chose an LCD tablet because I it provides a more intuitive way to record interactions with students. The hardware considerations for projects like this one are discussed in Section 3.1.

Chapter 3: Recording Technologies

In order to create a system to provide students with helpful recordings of educational interactions, I had to determine a good way to create these videos. In order to preserve the information from these interactions, I explored a number of different hardware and software technologies. The set of tools that I chose for this project might well be surpassed technologically in the near future, but the reasoning behind selecting various tools and the feedback from students in the user study both illustrate principles that could be helpful to anyone looking to design a similar system.

Many of the student-teacher interactions involve looking together at a piece of paper or a chalkboard while the teacher draws and explains. In order to record this kind of interaction, I chose to focus on the audio of the student-teacher communication and the drawing surface where the visual part of the explanation takes place. In order to provide this functionality, I needed recording software, video editing/processing software, and a computer-connected drawing surface that provided an interface as intuitive to use in an explanation as a chalkboard or piece of paper. I explored possibilities for each kind of tool before deciding on the eventual set that worked best, and the following sections discuss the merits of some of the notable technologies considered for the project.

3.1 Input Hardware

Finding an intuitive drawing surface was one of the key concerns in setting up a recording system. Versatility and portability were also important factors. I looked at a unified recording/interaction surface, two USB tablet input devices, and mobile tablet devices. While mobile tablet devices might be more portable and convenient in the future, it seems that the hardware and applications available to easily set up a recording system might not have been developed at the time of this project.

3.1.1 Smart Board

The SMART Board is an interactive whiteboard solution with bundled software. This was attractive as a potential solution because it provided a highly visible surface for student interaction. The SMART Boards provide support for user touch interaction to scroll through content larger than the board can display, and accepts writing input through special digital pens. Content is displayed on the board and pen input is captured from the board using SMART Technologies proprietary software. Projectors and cameras are used in various configurations across the different models to provide the interactive element.

It was somewhat difficult to obtain an exact figure for the cost of a SMART Board in any such system because the company only gives prices in individual discussions about total "solutions." Media releases from SMART Technologies provide some general figures for suggested retail value. The SMART Board 880 [3] offers support for simultaneous users through the "SMART Notebook 10.7 collaborative learning software" that is published by SMART Technologies. The suggested retail value for the board alone is $1,999 and for the entire system, including the projector, it is listed as $3899. Some of the newer products, such as the SMART Board 885ix interactive whiteboard system [4] provide advanced features for meetings, including integration with SMART conferencing software. The product announcement lists the suggested retail for this second solution as $6499, with conferencing software starting at an additional $5999.

The SMART Board did not seem like an ideal solution because of the bundled software limitations and the high price. The demo version software for the SMART Board seemed very well suited for displaying and interacting with images and prepared presentations, but it was incredibly unintuitive for displaying simple pen-input. The software offered many other features to integrate with student mobile devices like tablet PCs but the full-featured version of the software was unnecessarily expensive for the scope of this project.

3.1.2 Intuous 4

Wacom's Intuous 4 line of tablets provides a high level of pressure sensitivity and tilt sensitivity. These tablets are solid-color, horizontal, user-input devices that use a dedicated, wireless pen. The tablets attach to a PC through a single USB connection, and are compatible with Windows, Mac, and Linux. Like all Wacom tablets, the product comes with a few software licenses, but these are not by any means the only compatible tablet input software products. The Intuous 4 tablets start at around $200 for the smallest model.

I used one of these to produce a few videos before switching to the Wacom Cintiq. The experience of writing on a horizontal surface while viewing the result on the vertical screen in front of me seemed highly counterintuitive, and while I was able to adjust to a certain degree, I wanted to come up with a solution that was easier to learn to use for recording. The time investment needed to train other potential video authors to write on this tablet fluidly seemed impractical, and I also did not think it would be intuitive and natural enough to be a good replacement for a "piece of paper" when offering explanations to students.

Additionally, the Intuous 4 seemed to require more memory to process tablet input because of the higher degree of granularity afforded by the device. For instructors interested in producing tutorial videos in private, without engaging students directly, there are other, more basic, less expensive tablets available. The Wacom Bamboo tablet, is one of these simpler devices, and it is the type of tablet reportedly used by Mr. Kahn in producing videos for the Kahn Academy, mentioned in Section 2.3.

3.1.3 Apple iPads and Similar Tablets

Apple iPads are highly portable, lightweight tablet devices produced by Apple. Though the current leading product in their category, there are many similar alternatives. Highly mobile solutions for recording student and teacher interactions are very desirable, and I did look into using devices like this for doing so. Simply put, at the time that this system was assembled, there were no readily available ways to record screen activity on these devices, and even if there were, the hardware might not have been able to keep up with recording and processing tablet input at the same time. Future iterations of this type of device will almost certainly offer the kind of functionality that would be useful for recording student-teacher interactions.

3.1.4 Wacom Cintiq 21ux

The Wacom Cintiq 21ux provided the best interface for simulating the natural feeilng of drawing on a piece of paper while explaining a concept to a student. The Cintiq 21ux is a 21-inch LCD tablet. Essentially, it functions as a 21-inch monitor with the ability to accept tablet input with the same high-performance levels of the Intuous 4 mentioned in section 3.1.2. The Cintiq is a much larger device than the other non-LCD Wacom tablets or mobile devices like iPads, but the bulk was actually helpful in providing a large enough surface for drawing-based explanations, and the tablet-enabled LCD technology provide to be incredibly intuitive to learn to use. It is also important to mention that, while students were drawn to the large "fancy-looking" tablet/LCD, once an instructor started using it in an explanation, the process of using a pen to draw in something paper-like was so natural for students that they were not distracted by the technology and were able to proceed with the interaction more naturally.

3.2 Tablet Input Software

For the tablet devices that I reviewed in the course of this project, tablet input registers as a mouse movements. There are countless software products that can function as a canvas for user input. For this project, I wanted a software product with a user interface that was not too cluttered because this might be too distracting during recording and leave instructors spend more time interacting with the UI to select brushes or colors than necessary to help students. It was also important to find a product that produced clean, legible brush strokes so that instructors' writing would be visible during the recording process. In addition to displaying brush strokes, I wanted to be able to quickly add images to a background layer so that it would be easy to draw on existing course material in the form of images and PDFs, because this is a common method that instructors use to help to explain things to students.

3.2.1 OneNote

Microsoft OneNote is a product offered as part of the Office Suite. The product was not offered for Mac or Linux platforms at the time of this project. OneNote offers interesting features like note synchronization across devices, but this was not particularly important for the immediate goals of this project. OneNote seems better suited for individual note taking where the finished result is meant to be static. The user controls are not large enough to be easy to find during recording. The lack of cross-platform support also makes it a less desirable choice. OneNote does, however, offer excellent support for multimedia interaction between images and writing, but given the previous limitations, this was not the ideal choice.

3.2.2Corel Painter

Corel Painter was one of the bundled software options available with the purchase of the Wacom Intuous 4 and the Wacom Cintiq. The user interface is sufficiently intuitive, and there is very fine-grained control over brush size, granularity, etc. Unfortunately, the same features that make this a great tool for artistic input also make prone to using too much memory, which causes a lag in registering pen input on the screen when the recording program is also running. Because extensive graphics features were not as important as smooth writing during recording, this was not the ideal solution.

3.2.3Sketchbook Express

SketchBook Express is a free product produced by Autodesk. There is a more full-featured, paid version, SketchBook Pro, but the free version was suitable for tablet input during recording. The user interface is sufficiently intuitive and free of clutter. The program does not use too much memory, and thus capturing tablet input while recording does not lead to significant lag in displaying pen movement. Adding images is as simple as pasting them into the file, and the functionality for putting different images in different layers is highly intuitive and easily accessible. The artistic depth is not as extensive as it is in Corel Painter, but this is not as necessary for capturing simple instructive interactions. There are also free versions of this product available on mobile devices running Apple's iOS and Google's Android operating systems. This makes

SketchBook Express a promising candidate for use with more mobile solutions when these become more feasible.

3.3 Recording and Editing

The final part of the recording setup was to select software for recording screen activity and audio of student-teacher interactions and then editing that content before publishing it to the course media site. Several software solutions offered simple screen recording, but not all of them offered intuitive and effective editing functionality or a low price point. A few notable solutions that I explored in the course of this project are discussed briefly in the following subsections.

3.3.1 QuickTime X

QuickTime X comes free with Apple Snow Leopard. One major disadvantage is that this is the only operating system that is compatible with it. QuickTime X. This software has several appealing features for exporting videos easily to various formats and web locations like YouTube, but it seems better suited for recording webcam input. There is a desktop recording feature in QuickTime X, but it is not possible to limit the recording to a particular window or monitor. The editing features are also very limited compared to other options. Given these limitations, QuickTime X might not even be the best free option for the given recording application where screen capture is the most important feature.

3.3.2ScreenFlow

Telestream's ScreenFlow is a home screen casting product that allows users to record, resize the recording area retroactively, cut video and audio tracks, splice in new ones, add text elements, and export the finished product to QuickTime or Flash files on disk [5] . There is also support for publishing videos directly to YouTube. ScreenFlow was about $100 at the time of purchase, and provided very intuitive ways to record and edit the videos for the various instructors who participated in the study. ScreenFlow is primarily a Mac application, but it can be run on other operating systems. ScreenFlow has some nice features to reduce the memory footprint while running, which is helpful when the tablet input software is also using system resources. While there are comparable solutions, I chose ScreenFlow because it provided the desired functionality without costing more money than necessary to achieve that.

3.3.3Camtasia

TechSmith's Camtasia is a full featured recording and editing application that is more specifically focused on screen recording and creating polished screen casts. Like ScrenFlow, Camtasia's interface makes it very easy to resize the recording area after the fact, add captions, and export videos to various formats. At the time of purchase, Camtasia was more expensive than ScreenFlow, but as of May 2011, education discounts [6] are available that reduce the price from $149 to $99. Camtasia also offers advanced support for automatically zooming in on areas of interest for smaller screens like those on mobile devices. Camtasia is the tool of choice for many educational media publishers, including Salman Khan of the Khan Academy mentioned in Section 2.3. Overall, TechSmith seems to offer more software products for conferencing, and interacting with images [7] that integrate with Camtasia, and this might make it a better choice for applications that would need to make use of that. I chose ScreenFlow because they both offered similar functionality for my purposes, and ScreenFlow was less expensive at the time.

3.4 Final Design

A successful educational content delivery system depends on the ability to create useful content quickly. Producing educational videos depends a tablet device on the hardware side, and software for displaying tablet input, screen recording software, and video editing. Tablet input and screen recording are both computationally intensive, and it is important to choose a set of components that balances the load placed on the system during recording with the quality of the visual feedback from the tablet during recording.

The physical tablet and tablet input software also need to be easy to use. The hardware and software components used in the recording process were chosen to balance the computational load with the quality of output and ease of production. I chose the Wacom Cintiq 21ux as my input device because it was highly intuitive, and it worked with a wide variety of software for displaying input and recording it. For tablet input software, I chose SketchBook Express because it was lightweight, sufficiently intuitive, and provided good support for including media such as images or PDFs for background. Finally, for recording software, I chose ScreenFlow because it made the editing process sufficiently intuitive, provided necessary functionality and was less expensive than comparable products. Armed with this set of technologies, I set about building the content delivery system to share recordings with students. The architecture of this content delivery system is described in the next section.

Chapter 4: Content Delivery

After settling on a recording setup, I needed a way to share these videos with students. ScreenFlow had a nice feature for exporting videos to YouTube, and I did share a few videos publicly that way, but I mainly used it as a way to demonstrate recording approaches to others while building the website. For some purposes, a simple public forum like YouTube is probably sufficient, but for deployment in a specific class, it made sense to build a website to organize the videos in a more meaningful way. I created a site that required a username and password to login and view content because I wanted to track user activity and because I anticipated that students might object to having videos that included their voice being shared publicly.

I began with very little web programming experience, and decided to use a web framework to help with managing the underlying database. This provided a good starting point for building a basic video viewing site, and from there, I incorporated other web programming tools for formatting specific site views, manipulating the embedded movie objects, sending information to the server to record user viewing habits, and generating graphs to show staff members students' aggregated viewing patterns for each video. Finally, when it came time to grant access to all 150 students in 6.004, I set up a more robust server configuration for serving the large media files to many users simultaneously.

4.1 Modular Features

In this section, I describe some of the key components of the site's user interface. This is meant as a high level overview to motivate further discussion of more complicated user views and underlying functionality. A more in-depth description of entire user views from various pages can be found in Section 4.2. Section 4.3 covers the system architecture.

The code behind each of the modular components described in this section can also be reused to further extend existing features. The main user interaction features for the site listed are the embedded movie object, view history graphs for staff members, a list of comments with a submission box to add a new one, a button to select or remove a video as a favorite, and a sorted table for browsing content.

4.1.1 HTML Movie Div

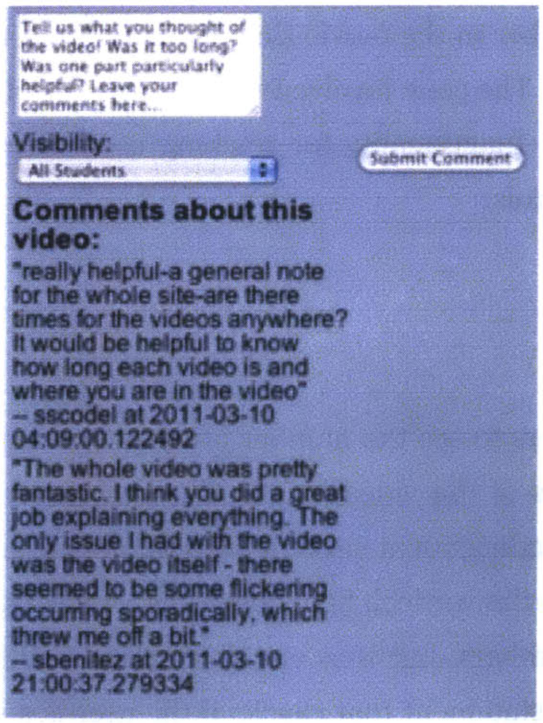
There are several JavaScript files and CSS profile definitions that define an embedded movie player. This player displays a QuickTime movie object and provides a navigational bar with start/stop buttons and a scrubber that one can move to change positions in the video. The value of this scrubber in the movie timeline is referred to in the code, and in this thesis, as the "playhead." The code involved in including the player object on a page in the system also includes functionality for tracking users' viewing behavior when interacting with the video controls.

4.1.2 View History Graphs

View history graphs allow staff members to see the number of users viewing the video continuously at evenly spaced intervals of the video length. The graphs are 2D plots with time on the horizontal axis ad the independent variable, and number of users watching continuously at that time plotted on the vertical, dependent axis. View history graphs are currently only visible to staff members, but they could be made visible to students by eliminating a simple conditional statement that checks if the user is a staff member within the `show_media` function in `views.py`. These graphs are generated using the Google Graphs [8] service. This process is described more in Section 4.2.2.3.

4.1.3 Comment Panel

The comment panel appears under the embedded movie object in the single video view, which is discussed in further detail in Section 4.2.1.3. Students can choose whether to make their comments visible to all students in the class, or to make them visible only to staff and the author. An example of this contrast in the staff and student views of comments for a particular video is shown in Figure 1. The list of comments automatically refreshes when users add new comments without forcing a refresh on the entire page, which would cause any video being played to restart.



Tell us what you thought of the video! Was it too long? Was one part particularly helpful? Leave your comments here...

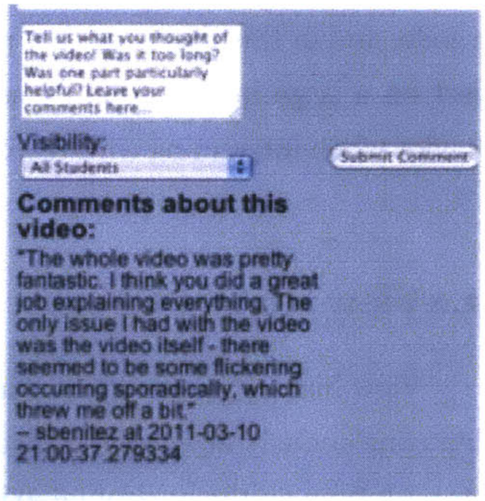
Visibility: All Students

Submit Comment

Comments about this video:

"really helpful-a general note for the whole site-are there times for the videos anywhere? It would be helpful to know how long each video is and where you are in the video"
-- sscodel at 2011-03-10 04:09:00.122492

"The whole video was pretty fantastic. I think you did a great job explaining everything. The only issue I had with the video was the video itself - there seemed to be some flickering occurring sporadically, which threw me off a bit."
-- sbenitez at 2011-03-10 21:00:37.279334



Tell us what you thought of the video! Was it too long? Was one part particularly helpful? Leave your comments here...

Visibility: All Students

Submit Comment

Comments about this video:

"The whole video was pretty fantastic. I think you did a great job explaining everything. The only issue I had with the video was the video itself - there seemed to be some flickering occurring sporadically, which threw me off a bit."
-- sbenitez at 2011-03-10 21:00:37.279334

Figure 1: (Left) Staff views of video comments. (Right) Student view of comments

4.1.4 Favorite Button

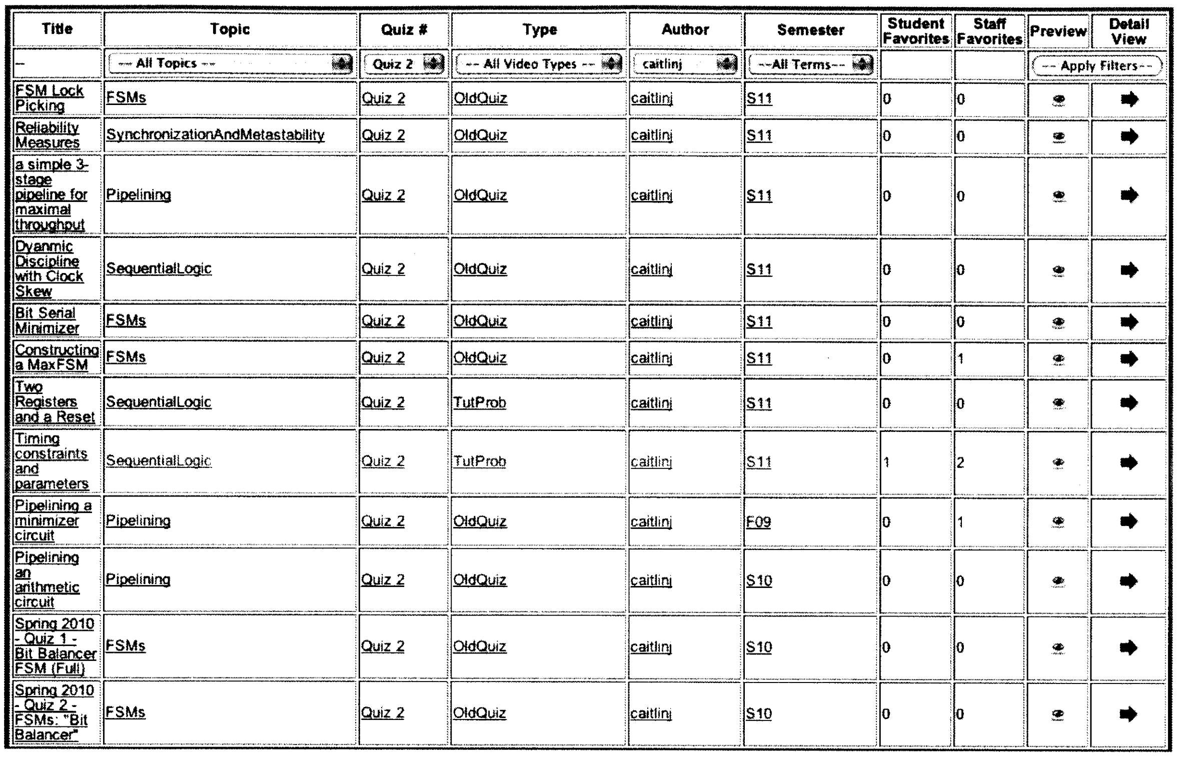
The favorite button also appears on the single video-viewing page. This allows students and staff easy access to toggling a video as a favorite without cluttering the video viewing area. There is a difference between student and staff favorites in the database and this is reflected in the media-browsing table that is discussed in Section 4.1.5. When a video has not yet been added as a particular user's favorite, then the button text will read "Add Favorite," and when it has been added previously, the button will read "Remove Favorite." The term "favorite" is used frequently throughout this thesis, and it refers to the database object created when a user interacts with this button.

4.1.5 Sorted Content Browsing Table

The sorted content browser appears on the main student-landing page, and it provides a quick way for students to find the set of videos most relevant to their needs. The full student landing page view is discussed in Section 4.2.1.2.

The browsing table allows students to filter videos by author, semester created, type of interaction in the video, and topic covered by the video. Students can also filter videos by the "quiz number," which means that they can choose to view the set of videos that staff members have chosen as study material for a particular exam. This feature was quickly added after it became apparent that students were very interested in browsing the videos to find study material for exams.

There are icons on the right of the row entry for each video that allow a student to preview the given video above on the same page, or to view the video in the separate, main video viewing page. Students can also choose to populate the entire list with only their favorite videos or the entire set of videos available to the class.



Title	Topic	Quiz #	Type	Author	Semester	Student Favorites	Staff Favorites	Preview	Detail View
--	-- All Topics --	Quiz 2	-- All Video Types --	caitlinj	--All Terms--			-- Apply Filters --	
FSM Lock Picking	FSMs	Quiz 2	OldQuiz	caitlinj	S11	0	0	👁	➡
Reliability Measures	SynchronizationAndMetastability	Quiz 2	OldQuiz	caitlinj	S11	0	0	👁	➡
a simple 3-stage pipeline for maximal throughput	Pipelining	Quiz 2	OldQuiz	caitlinj	S11	0	0	👁	➡
Dyanmic Discipline with Clock Skew	SequentialLogic	Quiz 2	OldQuiz	caitlinj	S11	0	0	👁	➡
Bit Serial Minimizer	FSMs	Quiz 2	OldQuiz	caitlinj	S11	0	0	👁	➡
Constructing a MaxFSM	FSMs	Quiz 2	OldQuiz	caitlinj	S11	0	1	👁	➡
Two Registers and a Reset	SequentialLogic	Quiz 2	TutProb	caitlinj	S11	0	0	👁	➡
Timing constraints and parameters	SequentialLogic	Quiz 2	TutProb	caitlinj	S11	1	2	👁	➡
Pipelining a minimizer circuit	Pipelining	Quiz 2	OldQuiz	caitlinj	F09	0	1	👁	➡
Pipelining an arithmetic circuit	Pipelining	Quiz 2	OldQuiz	caitlinj	S10	0	0	👁	➡
Spring 2010 - Quiz 1 - Bit Balancer FSM (Full)	FSMs	Quiz 2	OldQuiz	caitlinj	S10	0	0	👁	➡
Spring 2010 - Quiz 2 - FSMs: "Bit Balancer"	FSMs	Quiz 2	OldQuiz	caitlinj	S10	0	0	👁	➡

Figure 2: Browsing table to sort media according to key attributes

4.2 Website Views

There are two different sets of possible site interactions for staff members and students. Members of either group can manage their own passwords, view content, add comments, and list favorites. Staff members can also upload new video files, assign topics for chapters to create new TopicAssignment content, and view informative graphs showing user viewing habits for each video. In addition, site administrators have separate access to the Django admin interface for viewing all objects in the database directly.

4.2.1 Student Perspective

After logging in, the first thing a student sees is the main student-landing page described in Section 4.2.1.2. From there, students can also preview videos in the same page, or view them in a larger player on a new page that includes a comment box and favorite button.

4.2.1.1 Login / Change Password

The login page uses the site's CSS profiles described in Section 4.3.2, but the main view function that handles the form processing for authentication is done with included functions from the `django.contrib.auth` package. These view functions provide necessary variables to view functions that the programmer is expected to provide. The simple HTML templates `login.html`, `password_change_form.html`, and `password_change_done.html`, listed in Appendix Section B.4 were the templates that I wrote for this. The Django auth package takes care of all of the authorization and validation, and site administrators can still change any passwords with the convenient Django admin interface described in Section 4.2.3. The simple login page is shown here in Figure 3 the change password form is shown in Figure 4, and the confirmation page for a changed page is shown in Figure 5.

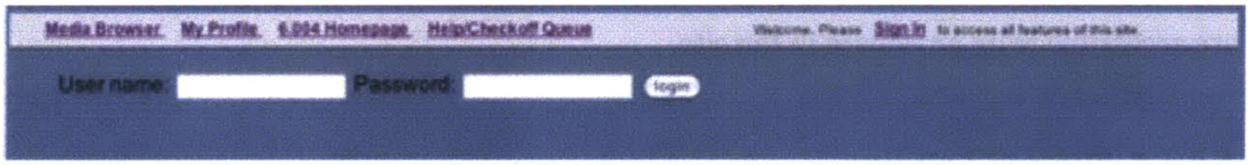
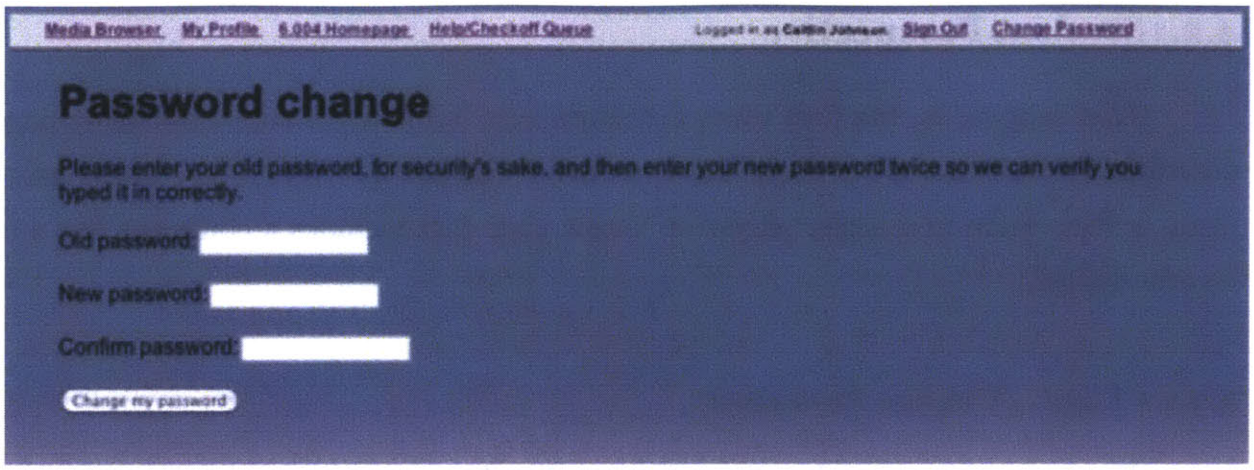


Figure 3: User login prompt


Media Browser My Profile 6.004 Homepage Help/Checkoff Queue   Logged in as Caitlin Johnson Sign Out Change Password

Password change

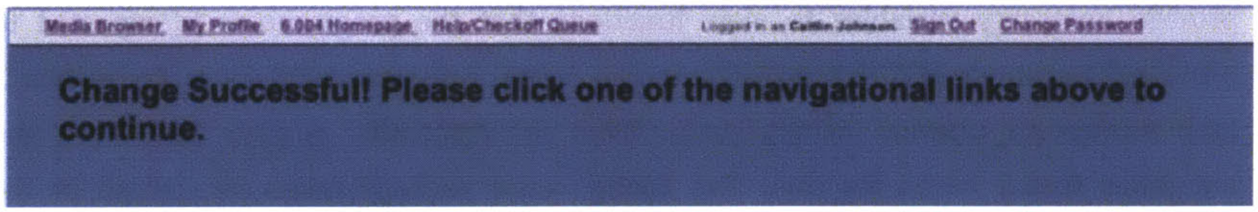
Please enter your old password, for security's sake, and then enter your new password twice so we can verify you typed it in correctly.

Old password:

New password:

Confirm password:

Figure 4: Password change form


Media Browser My Profile 6.004 Homepage Help/Checkoff Queue   Logged in as Caitlin Johnson Sign Out Change Password

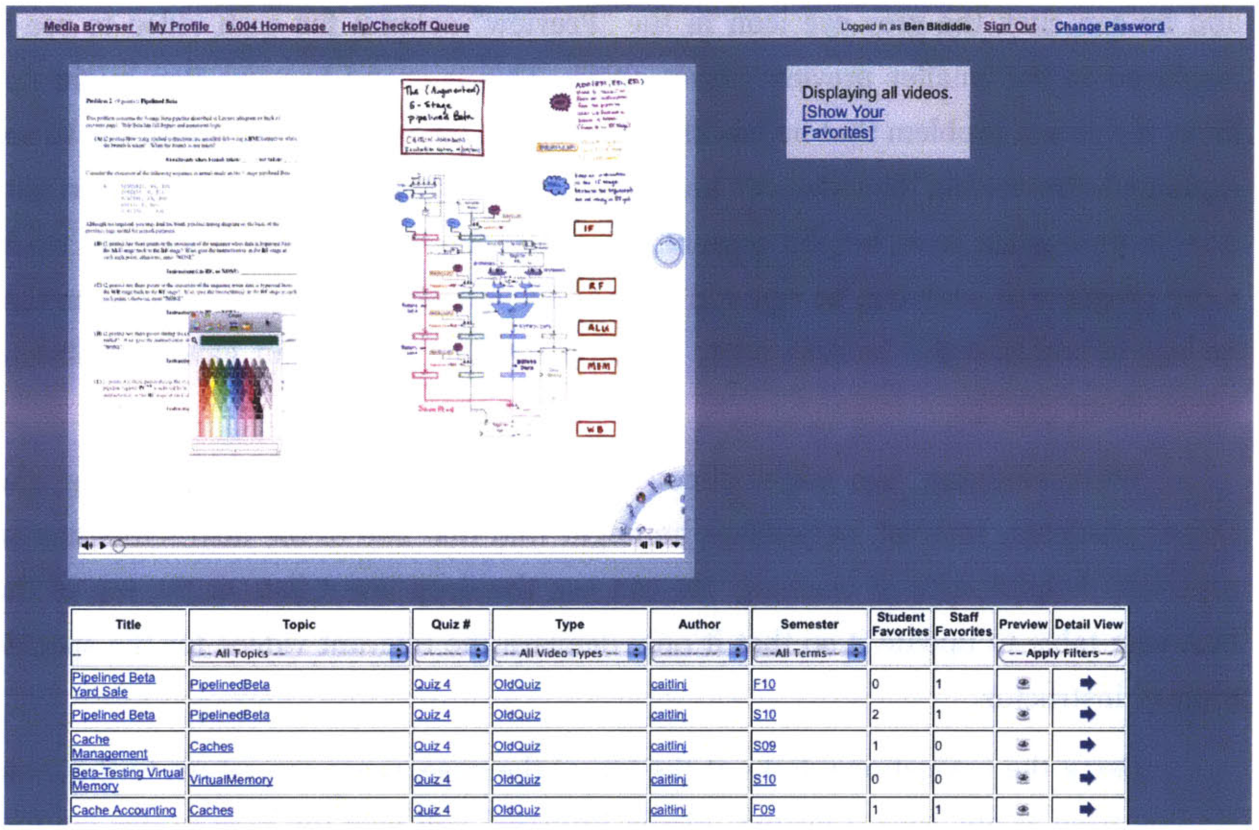
Change Successful! Please click one of the navigational links above to continue.

Figure 5: Password change confirmation

In these figures, one can see that each page has the basic masthead template and includes easily accessible links for students to login, logout, and change their passwords. This appears at the top of every page in the site, and thus, the login pages discussed here are accessible from anywhere in the site.

4.2.1.2 Student Landing

The main student-landing page provides a place for students to view all content on the site and keep track of their favorites. Figure 6 shows an example of a student-landing page. For a hypothetical student, Ben Bitdiddle.



Displaying all videos. [Show Your Favorites]

Title	Topic	Quiz #	Type	Author	Semester	Student Favorites	Staff Favorites	Preview	Detail View
--	-- All Topics --	------	-- All Video Types --	--------	--All Terms--			-- Apply Filters--	
Pipelined Beta Yard Sale	PipelinedBeta	Quiz 4	OldQuiz	caitlinj	F10	0	1	👁	➡
Pipelined Beta	PipelinedBeta	Quiz 4	OldQuiz	caitlinj	S10	2	1	👁	➡
Cache Management	Caches	Quiz 4	OldQuiz	caitlinj	S09	1	0	👁	➡
Beta-Testing Virtual Memory	VirtualMemory	Quiz 4	OldQuiz	caitlinj	S10	0	0	👁	➡
Cache Accounting	Caches	Quiz 4	OldQuiz	caitlinj	F09	1	1	👁	➡

Figure 6: Sample student landing page

At the top of the page is the usual masthead with course website and login links, the top left of the page shows the movie preview area, the bottom of the page shows the table that allows users to sort videos by their particular attributes, and the smaller box at the right contains a link that will change the list of videos in the sorted table to be either a student's favorites or all of the videos in the system.

In this example, the student, Ben, might be preparing for "Quiz 4." He might have come to the site to look for study materials. The most recently posted videos are in-lab explanations of old quiz problems that the instructor recorded when other students came to office hours and asked for help in working through these particular problems.

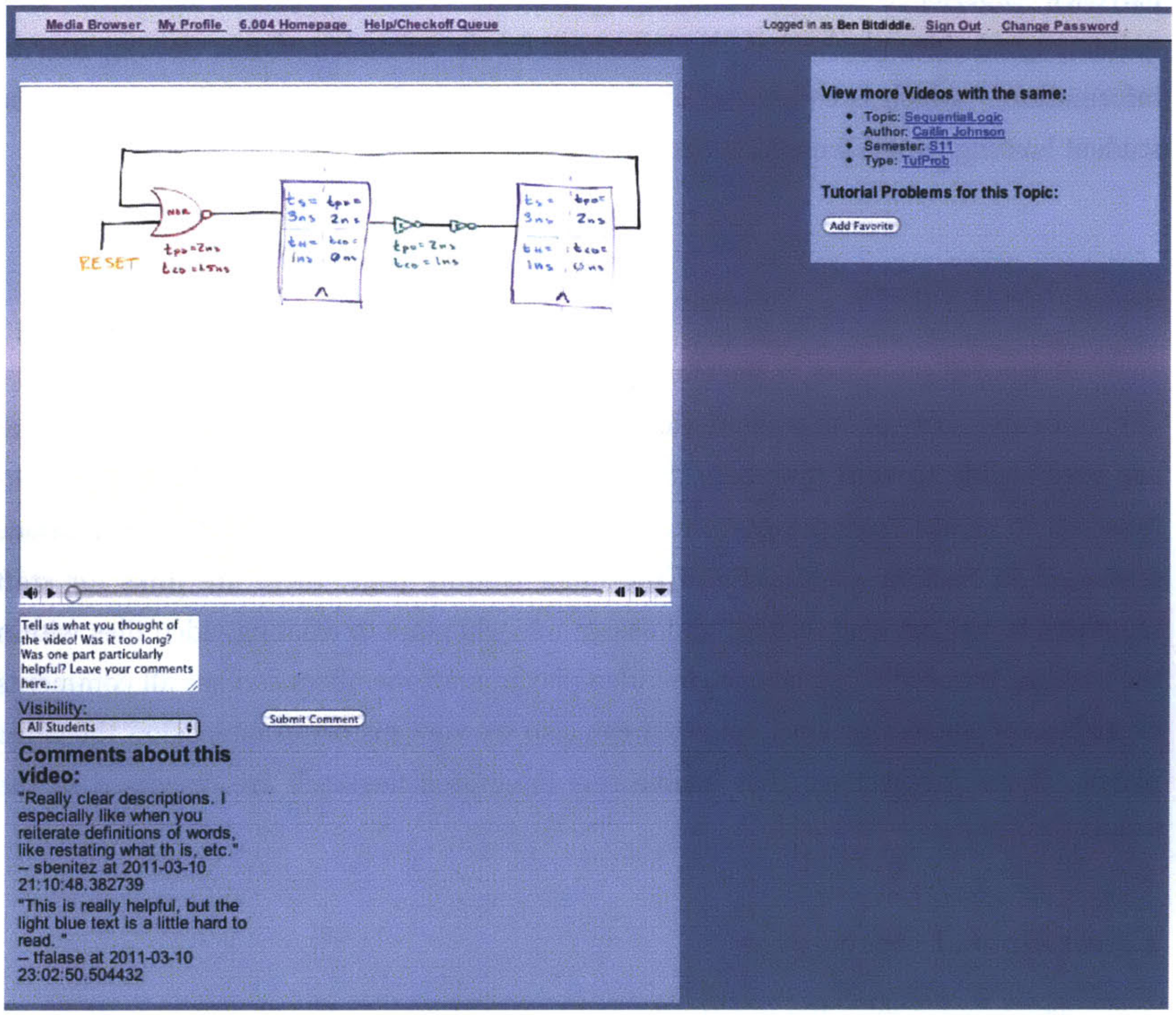
Ben can also see if other students have added any of the new explanations as favorites, and he can click the grey eye icon in the "Preview" column for each video to see it displayed in the smaller preview player above. This is useful because Ben might not know if this is a problem that he has already seen or if it's one that he would like to see explained. Previewing the video can give Ben a sense of the problem and help him save time in finding the most helpful content. Once Ben has viewed more videos and made more progress in studying, he can add videos as favorites, and then, the next time that he logs in and sees this landing page, he could click the "Show your Favorites" link to access them quickly.

While studying, Ben might discover that one topic is especially confusing, and this of particular interest, or perhaps he might find that one of the instructors has an especially helpful style of teaching. He can use the drop down lists at the top of the browsing table to update it so that it only displays the relevant videos for the specific topic or instructor.

Using the navigational links at the top of the page, users can return to the main landing page at any time. Clicking "Media Browser" will take the user to this landing page with all videos displayed in the table, and clicking "My Profile" will take the user to this page with only his or her favorites in the list. Clicking on the blue arrow icon, located in the "Detail View" column for each video, will take a user to the single-movie viewing page discussed in the next section.

4.2.1.3 Movie View

The single movie view page allows students to see a larger view of a particular video chapter. They can also see all comments that others have made publicly visible, and submit their own comments. An example of a single-movie viewing page with commentary, the comment submission form, and favorite button is shown below in Figure 7:



Tell us what you thought of the video! Was it too long? Was one part particularly helpful? Leave your comments here...

Visibility: All Students

Comments about this video:

"Really clear descriptions. I especially like when you reiterate definitions of words, like restating what th is, etc."
– sbenitez at 2011-03-10 21:10:48.382739

"This is really helpful, but the light blue text is a little hard to read. "
– tfalase at 2011-03-10 23:02:50.504432

A student like Ben Bitdiddle, who was introduced in the previous section, could make a comment public for the entire class, or he could select the option for sharing the comment only with "Staff Members and [username]" if he wanted to provide feedback to the staff or ask a question that he did not feel comfortable sharing with everyone else. As discussed in Section 4.3.3.3, the favorite button, included in the sidebar, and the comment submission form both use Ajax to send data to the server so that the user's playback experience is not disrupted while the student provides feedback. This uninterrupted video playback is also important for the collection of user viewing interval information – which is conducted on this page and in the movie preview pane in the student landing view discussed in Section 4.2.1.2.

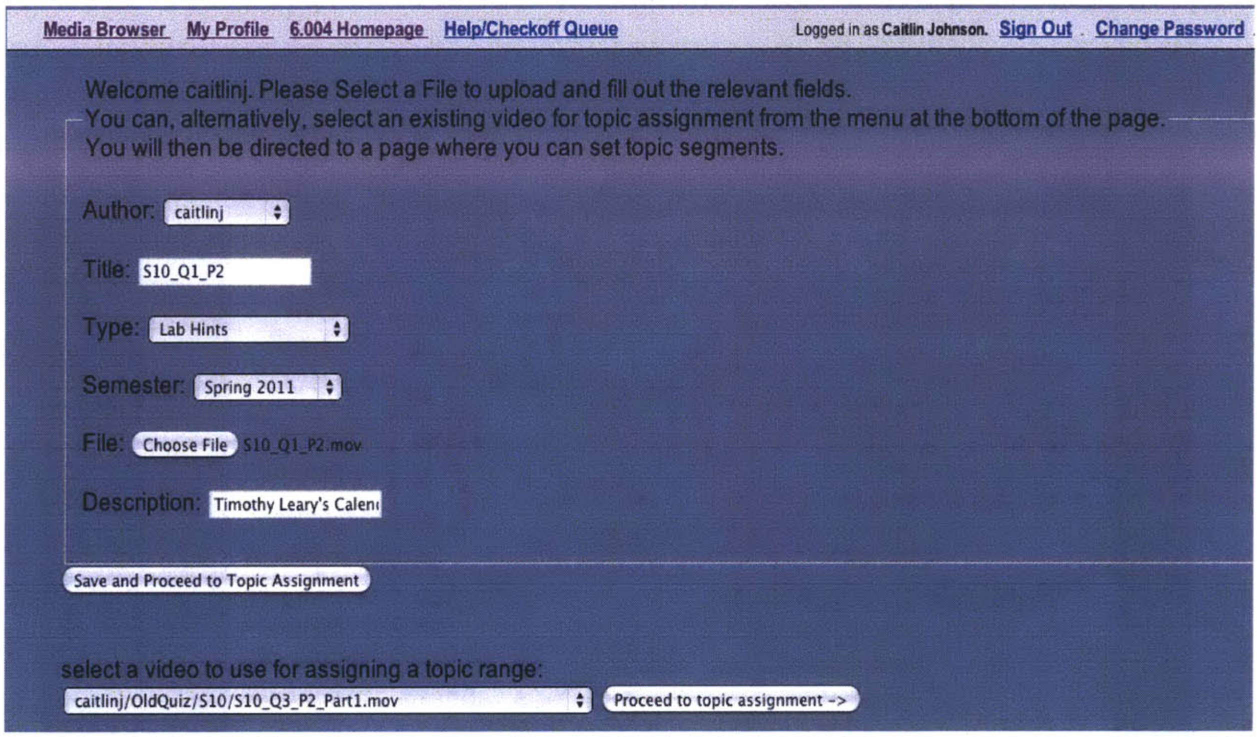
4.2.2 Staff Views

Several of the views in the site are available only to authenticated staff members. There are also parts of the student-viewable pages discussed in the previous section that are only visible to staff members. Staff members can upload videos, assign chapter intervals to create TopicAssignment objects, and access the separate Admin interface discussed in Section 4.2.3.2. From the main landing page, there are links for staff members to upload new content and assign new chapters to existing video files within the system. When viewing the single-video player, staff members can see all comments regardless of permission level, and they can also see view history graphs below the video player. These perspectives that enable this integrated feedback are discussed in the following sections.

4.2.2.1 Upload Video

This perspective allows a staff member to upload a video file, and specify basic parameters. This completes the creation of a PublicVideo object within the database, as discussed in Section 4.3.1.1, but it does not complete the creation of a TopicAssignment object; that is handled by the topic assignment page that the instructor sees after uploading a video to this form. If an instructor would like to add a new chapter to an existing video, he or she can select one form the drop down menu at the bottom.

This page is specified by the `upload_video` view function in `staff_views.py` as shown Appendix Section A.2.3 and the HTML template `upload_video.html`, shown in Appendix Section B.5.1. An example of the video upload form is shown below in Figure 8.


Media Browser My Profile 6.004 Homepage Help/Checkoff Queue Logged in as Caitlin Johnson. Sign Out . Change Password

Welcome caitlinj. Please Select a File to upload and fill out the relevant fields.
You can, alternatively, select an existing video for topic assignment from the menu at the bottom of the page.
You will then be directed to a page where you can set topic segments.

Author: caitlinj

Title: S10_Q1_P2

Type: Lab Hints

Semester: Spring 2011

File: Choose File S10_Q1_P2.mov

Description: Timothy Leary's Calen

Save and Proceed to Topic Assignment

select a video to use for assigning a topic range:
caitlinj/OldQuiz/S10/S10_Q3_P2_Part1.mov Proceed to topic assignment ->

Figure 8: Staff form for uploading a new video

4.2.2.2 Topic Assignment

The topic assignment page allows instructors to preview videos after uploading then, and then designate intervals within the video as chapters assigned to a particular topic. Instructors can move the playhead to the desired start of the chapter, and then click "Set to Now" under start time to set the HTML form value. The chapter end time is set the same way with the button under the "End Time" label. Instructors need to also assign a topic to the chapter, a subtitle that gives more information than the whole movie title, and also designate the quiz for which this video could be study material. The "quiz number" designation is more of a specific feature that I found particularly useful for the specific course in which this system was evaluated. Figure 9 shows an example of creating a topic assignment.

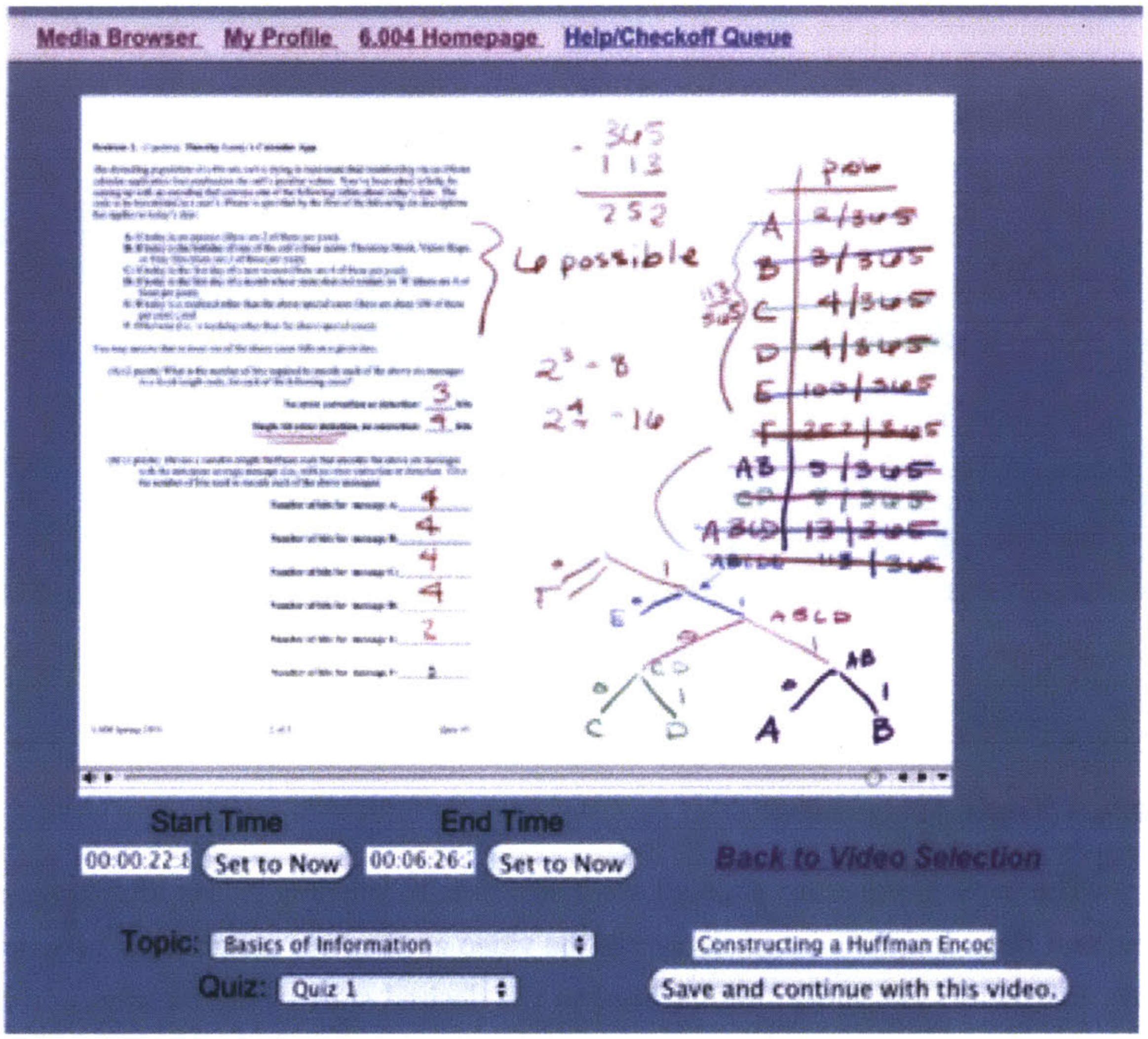


Figure 9: Topic assignment and movie preview

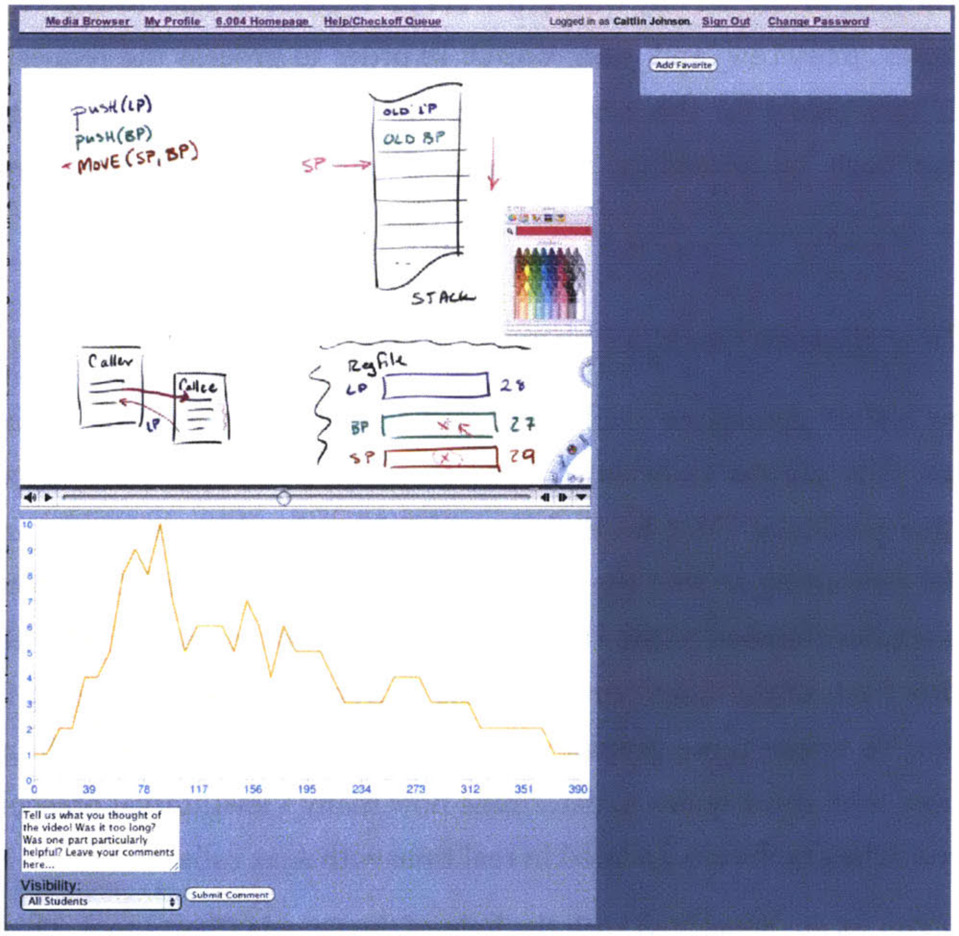
After setting the chapter start and end times, and the other relevant information, the author can choose to continue assigning chapters to this video file by clicking the "Continue with this Video" button, or he or she can choose to return to a list of all videos in the system to be able to create new chapters from them. The "Continue with this Video" button is the only one that submits the information to the server to create a new TopicAssignment object, though. To get back to the main media browser, an author can click the relevant links in the masthead that appears at the top of this page, as it does every page in the site.

Once an author saves the chapter, the information is sent through a standard HTML form post to the server, where the HTTP request is sent to the view function, `staff_views.preview_and_set_topic` in order to process the request and create a new database object for the new chapter. This view function is the same one that renders the page by providing relevant variables to the `movie_preview.html` template.

4.2.2.3 View History Graphs

View history graphs are available to staff members as part of the individual video player page. The graphs show how many students were in a state of continuously watching that particular video for each part of it. The view history graphs are displayed with careful formatting so that the axis scales alongside the QuickTime controller that provides a similar timeline of the video. In this way, instructors can move the playhead to easily view parts of the video corresponding to trends in student viewing for that part of the video. To create these graphs, I first divide the total video length into uniform intervals, and then run a query to determine how many ViewInterval objects cover each time interval. The graphs are updated in real time with Ajax calls.

The graphs are generated with the help of the Google Charts tool. This tool allows one to construct a URL with a specific format in order to render an image of a corresponding chart. The function `get_img_url` within `staff_views.py` goes through the many steps of constructing this URL. While the long URL string may seem complicated, the ability to construct a URL to retrieve custom images is vastly preferable to writing the software locally to create the charts, especially while in the process of developing a new system and determining the kinds of charts that one might want. One example of the entire staff video player view with a view history graph, dashboard, comment panel, and favorite button is shown in Figure 10. More discussion of how these graphs were used and interpreted in the user study can be found in Section 5.3.3.



Tell us what you thought of the video! Was it too long? Was one part particularly helpful? Leave your comments here...

Figure 10: An example of a view history graph

4.2.3 Django Admin

The admin application is a standard Django library that can be enabled with a single line in the `settings.py` file from the main application. This settings file also specifies the project database, so the admin application can be associated with a single database file. This allows it to provide a useful interface for managing objects within the database – including users. Fore more information on the Django web application framework, see Section 4.3.1.

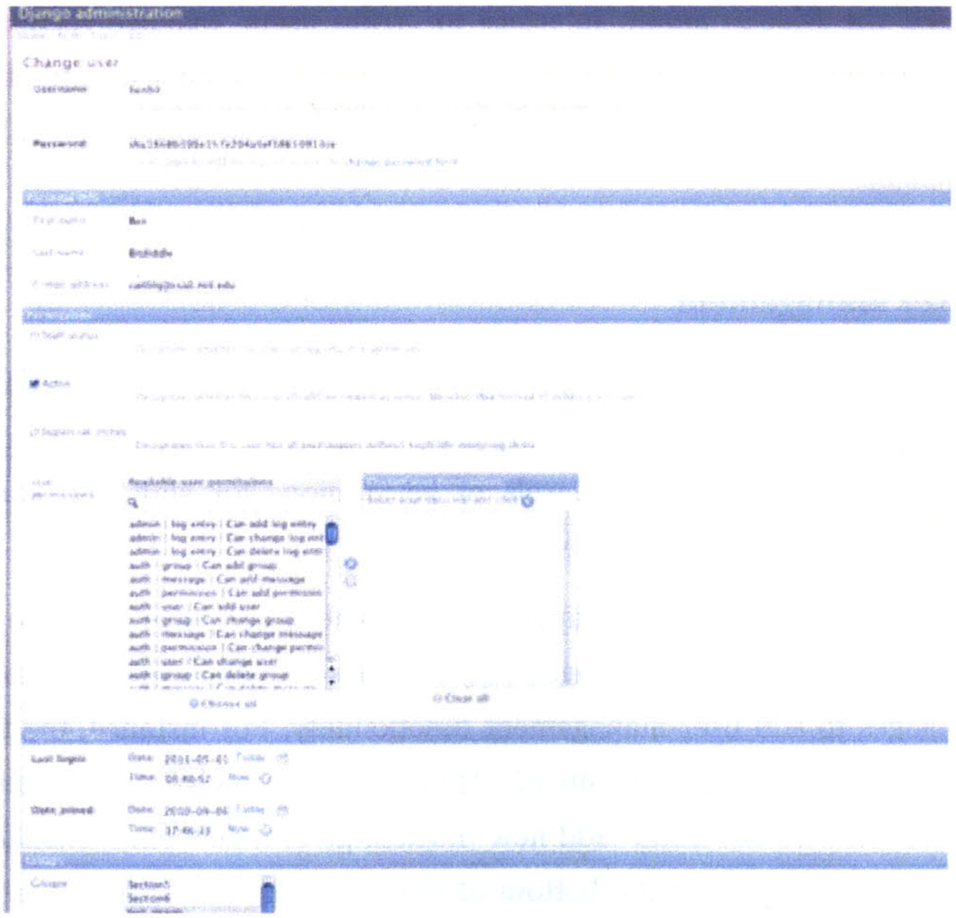
I used the Django admin interface to create users, manage section lists, and change passwords. I also used it to set up test data in the database, and change video information after the usual staff-upload process (described in Section 4.2.2) had already been completed.

4.2.3.1 User Management

The Django admin and auth packages provide useful functionality for managing users, setting permissions, and assigning users to groups. I found it very helpful for changing passwords, customizing levels of permissions for staff and students, and setting up groups for each recitation section.

The admin interface is less appealing in some other ways. Because it uses the built in auth package for user management, it is more complicated to try to extend the admin packages to add user management functionality. For instance, there was not a good way to simply add 150 students at a time. In order to give the students access to the site, I had to click the small "add new" button, go to a new page, change the group and permissions manually at the bottom of the page, and then repeat the process for each student. The admin interface also does not provide a great way to send email to all members of a group, as it seems that groups were more intended as a way to manage varying levels of permissions. I was able to write a script to send email to an entire list, but was a much less desirable solution than having standard email lists in any desktop mail application.

Despite these shortcomings, the admin interface provided a good start for user management, although it lacked ready interfaces for adding users in large batches and sending email to groups. An example of the Django user administration page is shown in Figure 11.


Django administration

Change user

Username: [illegible]

Password: [illegible]

First name: Ben

Last name: [illegible]

E-mail address: [illegible]

Staff status

✓ Active

Superuser status

User permissions: Available user permissions

admin | log entry | Can add log entry
admin | log entry | Can change log entry
admin | log entry | Can delete log entry
auth | group | Can add group
auth | message | Can add message
auth | permission | Can add permission
auth | user | Can add user
auth | group | Can change group
auth | message | Can change message
auth | permission | Can change permission
auth | user | Can change user
auth | group | Can delete group

Choose all

Clear all

Last login: Date: 2011-05-[illegible] Time: [illegible]

Date joined: Date: 2010-09-06 Time: [illegible]

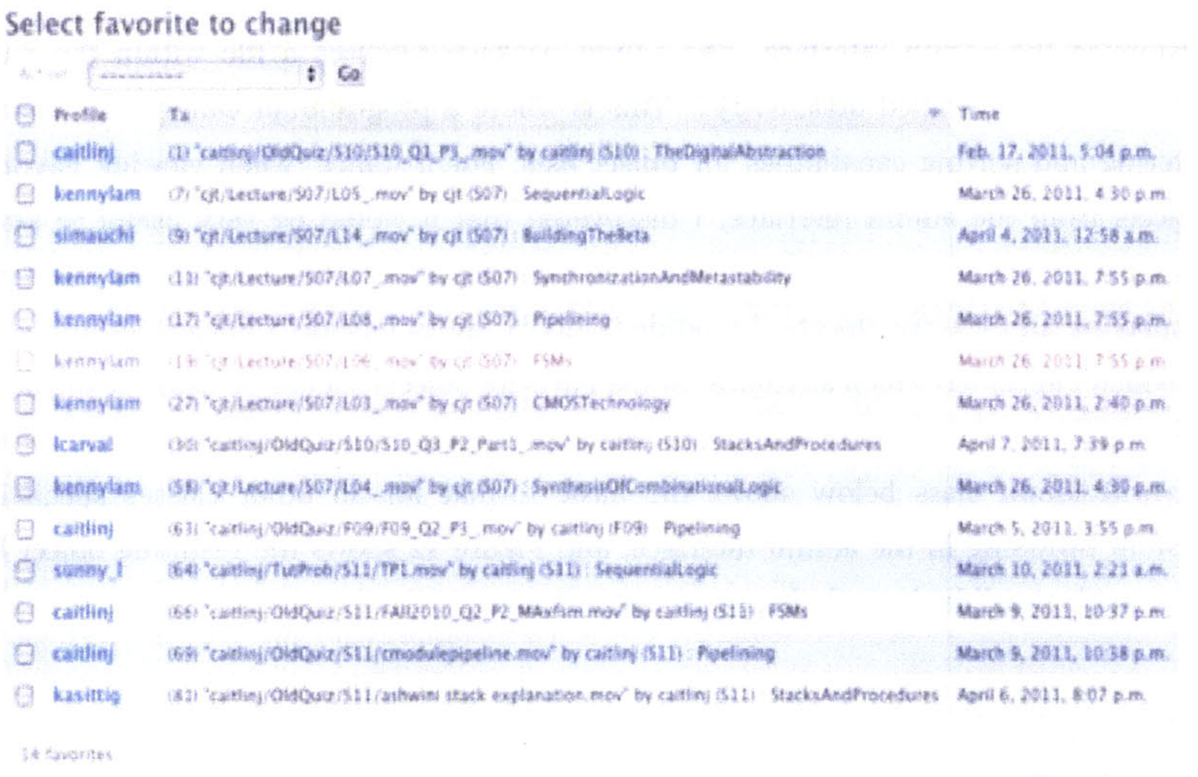
Groups: [illegible]

Figure 11: User administration

4.2.3.2 Object Browsing – Sorting and Filtering

For models that I defined in my own application, it was fairly straightforward to customize the admin interface. The admin package requires a file within the main project directory called `admin.py`. This is where a programmer can specify custom filtering and sorting capabilities for object lists. For instance, when viewing Favorite objects from the admin interface, I discovered that it would be very useful to know which ones were more recent. In order to add this as a sorted column in the table displaying all Favorite objects, I modified `admin.py` to include a FavoriteAdmin class in which I specified which attributes of the Favorite objects should be listed in the table, and which ones should be a way to sort the objects in the table. The simple code for the FavoriteAdmin class below shows the basic format for all other classes specifying custom browsing in the admin interface, and Figure 12 shows the resulting object list view made possible through these customizations.

```
class FavoriteAdmin (admin.ModelAdmin):
    list_display = ('profile', 'ta', 'time')
    list_filter = ('profile', 'ta', 'time')
```



Select favorite to change

Action: --------- Go

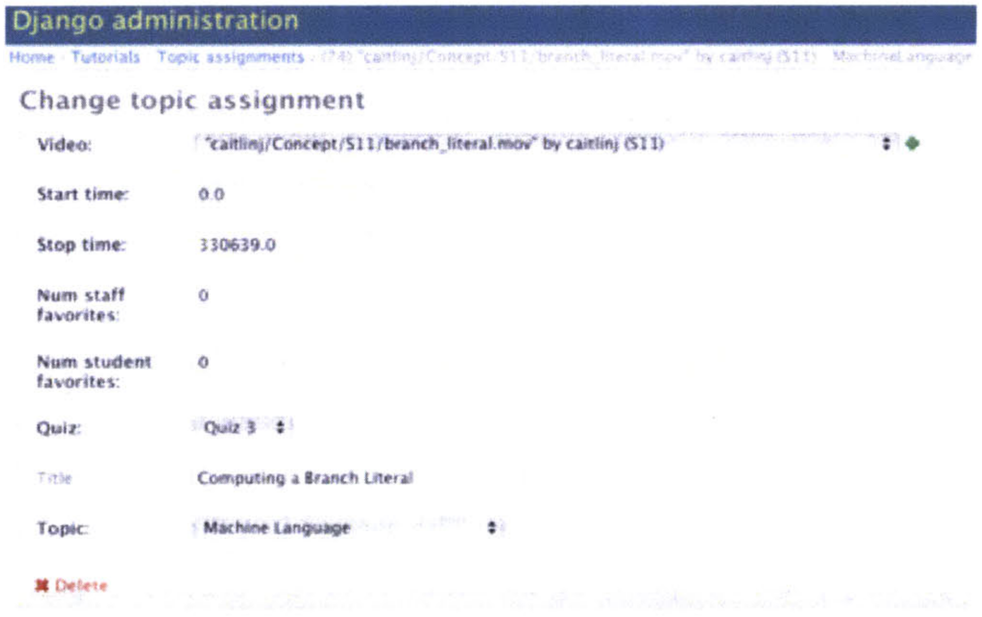
Profile	Ta	Time
caitlinj	(1) "caitlinj/OldQuiz/S10/S10_Q1_P3_.mov" by caitlinj (S10) : TheDigitalAbstraction	Feb. 17, 2011, 5:04 p.m.
kennylam	(7) "cjt/Lecture/S07/L05_.mov" by cjt (S07) : SequentialLogic	March 26, 2011, 4:30 p.m.
simauchi	(9) "cjt/Lecture/S07/L14_.mov" by cjt (S07) : BuildingTheBeta	April 4, 2011, 12:58 a.m.
kennylam	(11) "cjt/Lecture/S07/L07_.mov" by cjt (S07) : SynchronizationAndMetastability	March 26, 2011, 7:55 p.m.
kennylam	(17) "cjt/Lecture/S07/L08_.mov" by cjt (S07) : Pipelining	March 26, 2011, 7:55 p.m.
kennylam	(19) "cjt/Lecture/S07/L06_.mov" by cjt (S07) : FSMs	March 26, 2011, 7:55 p.m.
kennylam	(27) "cjt/Lecture/S07/L03_.mov" by cjt (S07) : CMOSTechnology	March 26, 2011, 2:40 p.m.
lcarval	(30) "caitlinj/OldQuiz/S10/S10_Q3_P2_Part1_.mov" by caitlinj (S10) : StacksAndProcedures	April 7, 2011, 7:39 p.m.
kennylam	(58) "cjt/Lecture/S07/L04_.mov" by cjt (S07) : SynthesisOfCombinationalLogic	March 26, 2011, 4:30 p.m.
caitlinj	(63) "caitlinj/OldQuiz/F09/F09_Q2_P3_.mov" by caitlinj (F09) : Pipelining	March 5, 2011, 3:55 p.m.
sunny_l	(64) "caitlinj/TutProb/S11/TP1.mov" by caitlinj (S11) : SequentialLogic	March 10, 2011, 2:21 a.m.
caitlinj	(66) "caitlinj/OldQuiz/S11/FAll2010_Q2_P2_MAxfsm.mov" by caitlinj (S11) : FSMs	March 9, 2011, 10:37 p.m.
caitlinj	(69) "caitlinj/OldQuiz/S11/cmodulepipeline.mov" by caitlinj (S11) : Pipelining	March 9, 2011, 10:38 p.m.
kasittig	(83) "caitlinj/OldQuiz/S11/ashwini stack explanation.mov" by caitlinj (S11) : StacksAndProcedures	April 6, 2011, 8:07 p.m.

14 favorites

For things like Favorite, Comment, or ViewInterval objects, this kind of custom sorting and attribute display made it easier to get a general idea of user interaction with the site. The information is still provided in the form as a list of database entries, though, and this is certainly less informative than other forms of feedback like the view history graphs mentioned in Section 4.2.2.3, The sorted list from the admin interface can, however, give instructors a rough view of current trends when there are not better reporting mechanisms, and can also give instructors and developers a better idea of the kinds of reporting that could be included to improve student-to-staff feedback in the system.

4.2.3.3 Modifying and Correcting Object Data

After viewing sets of objects in the database, as described in the previous section, site administrators can click one of those for a detail view. This provides a way to not only view an object's attributes, but to also modify them. This is particularly useful for doing things like changing the quiz number associated with a particular video without removing the existing data associated with the video, like user comments, etc. This is a good way to quickly keep information up to date in the midst of changing course curricula between semesters, and it is also a good way to correct errors after the fact. An example of an object-editing page for a TopicAssignment object is shown in Figure 13.


Django administration
Home › Tutorials › Topic assignments › (74) "caitlinj/Concept/S11/branch_literal.mov" by caitlinj (S11) › Machine Language
Delete

Change topic assignment

Video:	"caitlinj/Concept/S11/branch_literal.mov" by caitlinj (S11)
Start time:	0.0
Stop time:	330639.0
Num staff favorites:	0
Num student favorites:	0
Quiz:	Quiz 3
Title	Computing a Branch Literal
Topic:	Machine Language

Figure 13: Modifying object properties and adjusting foreign-key relationships

4.3 Website Framework

The functionality behind the website views described in Section 4.3 is provided with a python-based web application framework, an underlying database, and some client-side scripting to facilitate collecting feedback from users. The rest of this section describes the architecture of these system components.

4.3.1 Django

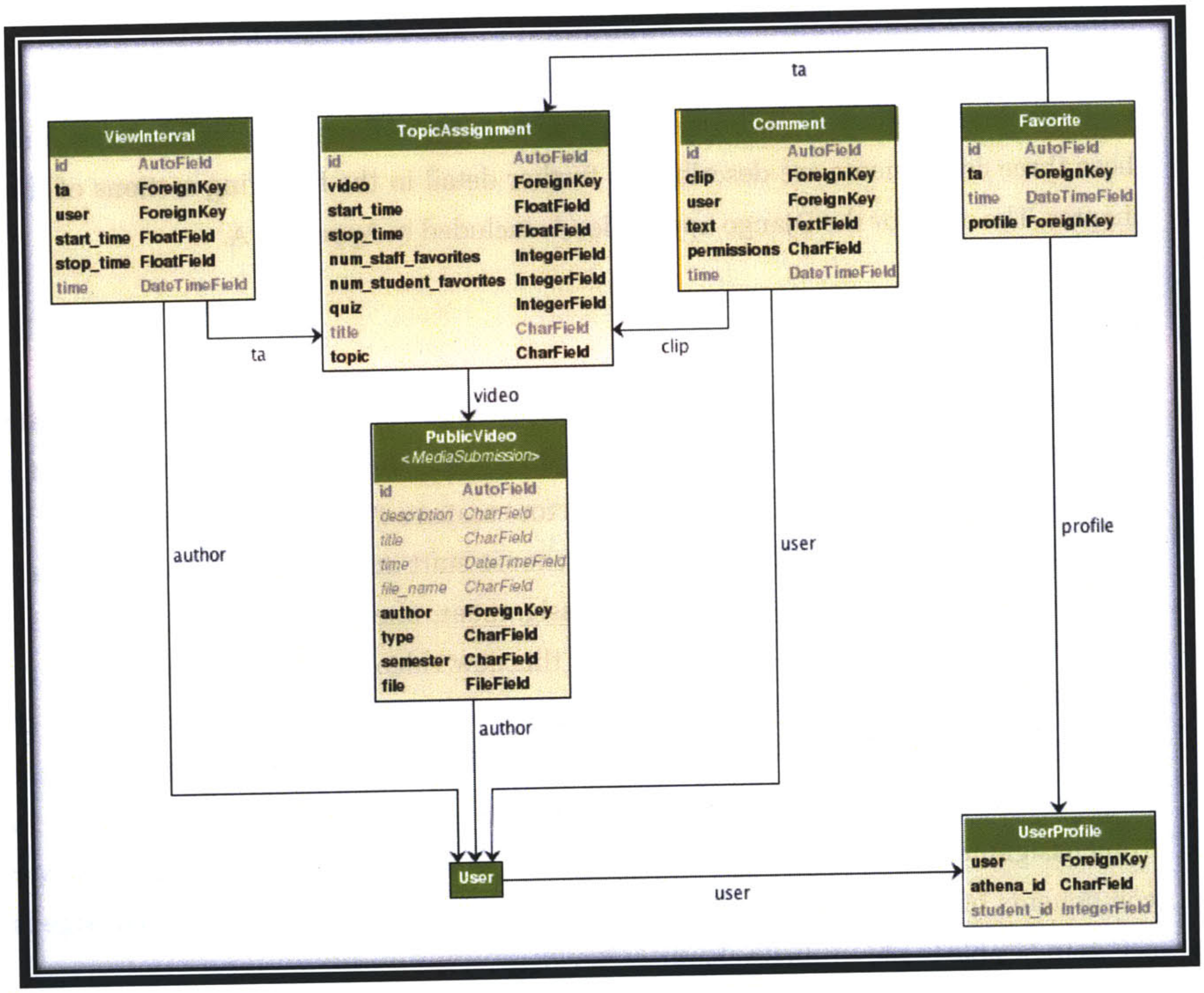
Django is an open-source web application framework written in Python. Django projects follow the "model-view-controller" (MVC) pattern. The "model" file allows a programmer to set up database tables automatically by specifying classes. The type of database can be specified in a separate settings file. Django has built-in support for PostgreSQL, MYSQL, and SQLite, among others. I used SQLite for developing this application, but rarely had to manipulate the database tables directly, as the Django framework provided much more intuitive ways to manage the information in the database.

A standard Django application includes a few key files that set up the basic model-view-controller architecture and help a user manage the application settings. The first of these is a file called `models.py`, in which one specifies the objects to be stored in database tables in python classes. The second is a file called `views.py`, and this is the primary place for specifying the behavior of the "controller" part of the MVC architecture. This file is the typical place to define functions to provide custom data to html templates, and to process incoming HTML requests to create or change the model data, if necessary. The fourth essential file is `settings.py`, in which an application creator specifies basic application settings, including relevant file paths, and the type of database to be used. The fifth file is `urls.py`, which provides a list of all URLs within the application, and the relevant function to render the html page for each one. Finally, the standard "view" component of the MVC pattern is realized by a set of html templates that are rendered with custom content by the aforementioned functions in `views.py`.

All files are included in the specific data models, view-rendering functions, and html templates together describe the basic functionality of the tutorial video system, and these three components are described in further detail in the following sections of this chapter. The code for the Django application is included in Appendix A.

4.3.1.1 Models (Model)

The main models in the database are depicted in Figure 14. Each User can create a PublicVideo object as an author, comment on a video they view, view a segment of a video, or add it as a favorite. When an instructor creates a PublicVideo and uploads it to the site, he or she is then taken to a page to choose start an end times to create at least one TopicAssignment. In creating a TopicAssignment, one must also choose the topic from an enumerated list and assign a title for this new video chapter. The object is called "TopicAssignment" because it is, at a basic level, the assignment of a topic to a pair of times. In terms of use, it is essentially a chapter object, and throughout this thesis, I frequently use the term "chapter" to in place of "TopicAssignment." After a TopicAssignment is created, this is the object that is used in the media player components of the pages, instead of the underlying PublicVideo, because each page is set up to be able to play a single chapter of a video.



The User object is borrowed from `django.contrib.auth.models`. This event triggers a function called `make_profile`, located near the end of models.py, which instantiates a corresponding UserProfile object for each unique User. When a User marks a video as a favorite, the request sent to the server will trigger a function to create a new Favorite object and associate it with the User's UserProfile.

A User can also be associated with a TopicAssignment through a Comment object. A Comment object also contains a permissions field, which comes from one of the `COMMENT_PERMISSION_CHOICES` in `enums.py`. Currently, there are only two levels in use, and those are "staff" and "all students." The staff level allows only course staff members and the comment's author to view it in the list of comments below the TopicAssignment in the standalone player page. The other level of permissions allows any student with login access to the videos to see the comment. This could be easily extended to accommodate students who may wish to leave a comment visible to those taking the course in the same semester, or students in their recitation section, etc.

A User does not have to mark a video as a favorite or leave a comment in order to leave a record of interaction with it. When a student plays a video for a continuous stretch of time, the start and end times of this are recorded as a ViewInterval. Saving these ViewInterval objects allows staff members to see the viewing behaviors of individual students through the Django admin interface and to see the number of students who were "tuned in" and watching each part of a video through a view history graph, as shown in Section 4.2.2.3.

4.3.1.2 Views (Controller)

Each URL included as part of the site in `urls.py` points to a view function in `views.py`, `student_views.py`, or `student_views.py`. Some of the URLs are in the form of regular expressions, allowing variables to be passed from the arguments in the URL. The full specification for this can be found in A.3.1. For the purposes of this explanation, an example is shown here:

In `urls.py`:

```
(r'^view_history/(?P<ta_id>\d+)/$',
staff_views.display_interval_views)
```

In `staff_views.py`, the function takes `ta_id` as an argument:

```
def display_interval_views(request, ta_id):
```

The functions in `views.py`, `student_views.py` and `staff_views.py` take in these variables and use them to extract relevant information from the database and then assign the values from the database to new names in a dictionary. This dictionary provides a context with which the to populate a html template by calling the Django shortcut method, `render_to_response`:

```
context = {
    "ta_start" : ta.start_time,
    "ta_id" : ta_id,
    "user": user,
           "ta_stop" : ta.stop_time,
    "selected_ta" : ta,
    "img_url" : img_url
    }
template = "display_interval_views.html"
return render_to_response(template, context)
```

Once the template has been rendered, and the html source provided to the user's browser, the user might initiate a POST request by interacting with the page to, for example, add a video as a favorite. These requests are sent to the function specified by `urls.py` for that URL. In some cases, this is the same function that rendered the template to begin with, but in the example of adding a favorite, it is not. The "Add Favorite" button could appear in many places, and the function `favorite_post` in `views.py` will handle the request to add or remove a favorite for a user and the relevant TopicAssignment object.

4.3.1.3 Templates (View)

The template system in Django allows for a programmer to use some HTML files to specify the layout of others. This makes it easy to use formatting tools like CSS profiles, as described in Section 4.3.2, but it can also make the behavior of the template-rendering a little bit confusing because of the number of files that are involved with rendering a single HTML view. Figure 15 provides a visual depiction of the template components involved in rendering the single chapter-viewing page that is described in further detail in Section 4.2.1.3.

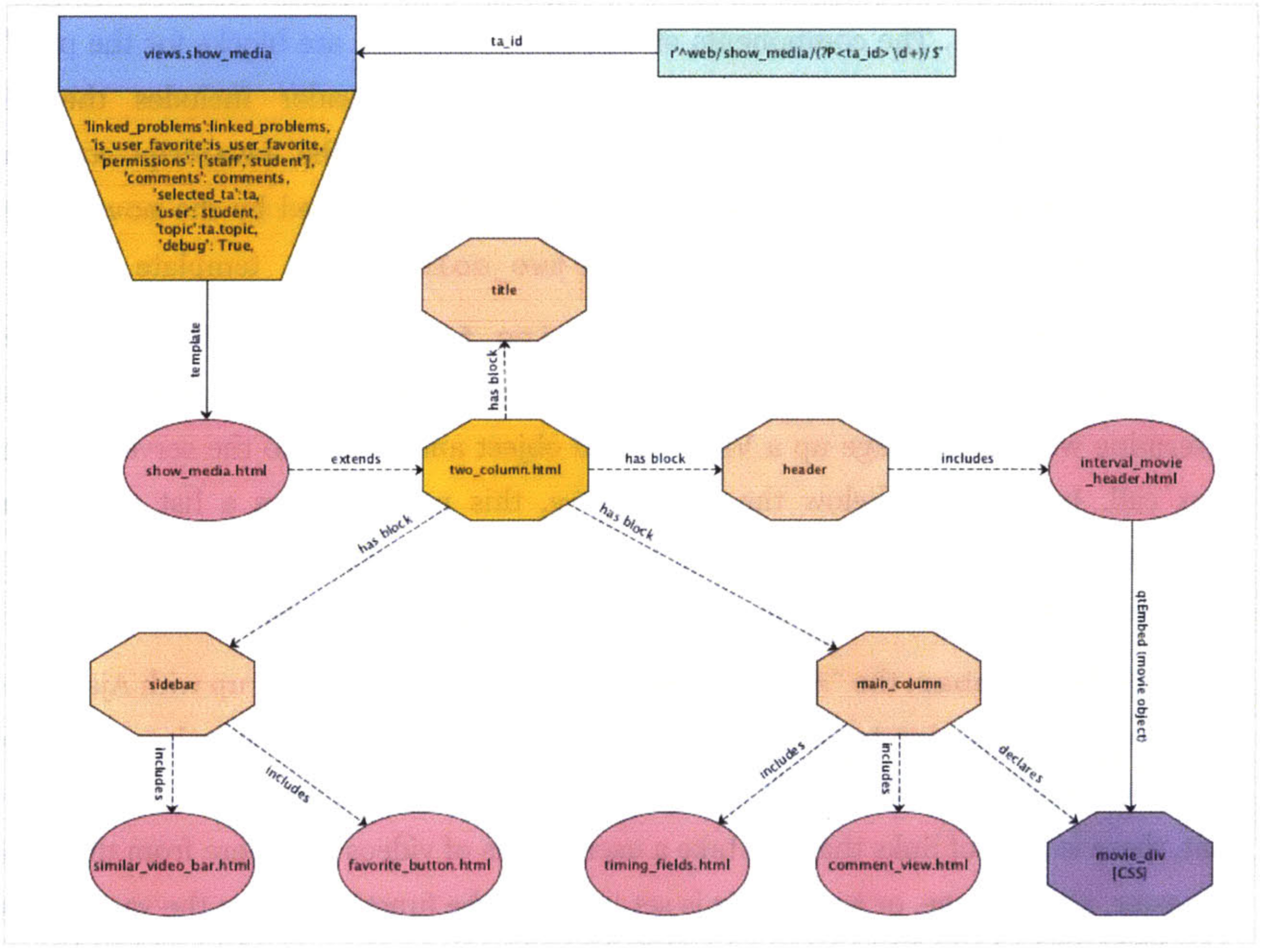


Figure 15: Template files involved in rendering a single-movie viewing page

In this example, the URL configuration file, `urls.py` contains a dictionary entry for the regular expression string `r'^web/show_media/(?P<ta_id>\d+)/$'` that points to the function `views.show_media`, and passes the parameter `ta_id` as an additional variable beyond the standard HTTP request object that is passed to every view rendering function. This function will produce a set of variable mappings with which to render the template (shown in the trapezoidal section below the rectangle containing the function name in the figure). This dictionary is passed to the `show_media.html` template, which extends a base template called `two_column.html`. The components of `two_column.html` are blocks for the page's `title`, `header`, `main_column`, and `sidebar`. The header includes the file `interval_movie_header.html`, which contains a lot of the JavaScript code for setting up the embedded movie object, which is then instantiated by the `movie_div` html field within the `main_column` part of the `two_column.html` template.

The `main_column` also contains `timing_fields.html`, which populates several hidden html fields used to monitor the play head position in the movie object to determine when to package up a ViewInterval object and send it to the server with an Ajax call. Immediately below the `movie_div`, this page displays a list of student comments, filtered by permissions, and text area and Ajax-enabled button for adding a new comment to the list. This is all set up within comment_view.html.

In the `sidebar`, the "Add as a Favorite" button is similarly set up with Ajax to be able to send information to the server and also to change the text if a user has added the video as a favorite already. This is set up within `favorite_button.html`. Right above that, there is a set of links that will take a user to lists of videos that were from the same semester, author, type, or topic. This is set up when the function passes the variables to the code in `similar_video_bar.html`.

This diagram is by no means complete. The standard navigational panel that appears on every page of the site, and contains links to the main page, course homepage, and lab help queue, is set up in the `header`. The `comment_view.html`, `favorite_button.html`, and `interval_movie_header.html` files all contain JavaScript code that sends data to other post handling functions in `views.py`, and then changes the content of the loaded web page without refreshing the entire page when the code, running on the client's machine, receives confirmation from the server that this request has been processed successfully. This is discussed in further detail in Section 4.3.3.3.

4.3.2 Cascading Style Sheets (CSS) and Base Templates

The basic layouts for all pages within the site are managed with Cascading Style Sheets, in `usersite.css`. Using CSS allows the programmer to define style attributes in a single location for types of HTML `<div>` items and classes that can be reused throughout the pages of the site. Unique items only exist once within a page and are identified by their `name` attribute within the HTML code. Classes provide general descriptions for more general types of objects, like links, paragraphs, etc., and are identified within the HTML code with the `class` attribute.

4.3.2.1 Base

The base template sets up the page header and the color scheme for all pages in the site. Every html template inherits from `base.html`, which is how the `usersite.css` style sheets are loaded in to every page. The `base.html` file includes a header block for JavaScript functions, the `masthead.html` file, and a main `content_container` class. The masthead lays out several `mast_link` items in a navigational bar, including a link to the main landing page, the external course website, a login link, and a link to a page for changing the password. The formatting for the masthead division and the `mast_link` class are both specified in `usersite.css`. The `content_container` is the wrapper for all other variable content on each page.

4.3.2.2 Two Block

Figure 15 in Section 4.3.1.3 shows an example of the rendering process for an instance of the two-block template. Section 4.2.1.3 describes the main use of this template within the system – the individual video player with comment views, and a favorite button. The template is set up by the file `two_column.html`. These sections are divided into two main blocks, the `sidebar`, and the `main_column`.

The `usersite.css` file contains setup information for the `browse_sidebar` and `sidebar_content` classes that define the outer region for the smaller right block, and the formatting for the internal content. This small block is meant to provide a set of navigational links for quick access to related content.

The `main_column` formatting, as defined in `usersite.css` fixes the position and width of this field so that it does not overlap the sidebar and so that both remain easily visible. In the `two_column.html` template's use in its extension in `show_media.html`, the `movie_div` and `comment_box` classes appear within the `main_column`. The formatting for these two classes is also laid out in `usersite.css`.

4.3.2.3 Three Block

The other main user perspective within the site is the main landing page, described in Section 4.2.1.2. The main CSS file, `usersite.css`, provides definitions for classes and unique fields in the `content_container` for `three_block.html`. This template reuses the `main_column` and `browse_sidebar` wrappers to display a box at the top left and a smaller sidebar at the right. This template also includes a `bottom_block` box that is wider and fills the width of the page below the other two blocks.

In the main landing page where this layout is employed, the `main_column` contains a preview of a selected video, the `browse_sidebar` contains login information for the student, and provides a link to toggle the query set for the table in the `bottom_block` as being the student's "Favorite" videos, or simply all the videos. This `bottom_block` includes a table listing the video-chapters' title, semester, author, type, topic, number of staff favorites, number of student favorites, and the quiz number that covers the material. The videos can then be sorted or filtered with these attributes. Once a student finds a video to watch, then he or she can either click a link to visit the single-video player with full comment detail, or click a link to preview that video in the smaller box above the table on the same page in the `main_column` block.

4.3.3 JavaScript

JavaScript files included in the page headers serve to gather data from the embedded QuickTime, set up the movie object, and send information about page events to the server. In order to send information to the server without requiring the page to refresh, I use the jQuery plugin to send AJAX (Asynchronous JavaScript with XML) requests to the server.

4.3.3.1 QuickTime

The general JavaScript library for QuickTime, `AC_QuickTime.js` is available for free download from Apple. The file `common_quicktime_methods.js` contains some additions to this library that proved useful for setting up the embedded QuickTime object. The most notable of these is `set_display_area_to_fit_movie(),` which formats the display area of the movie for playing in a `movie_div` on either of the main content viewing pages. Utilities like `playhead_position()` and `time_scale()` are convenience functions to reduce the amount of typing to get values for current attributes of the movie object, which are then used in `interval_methods.js`, and the JavaScript methods within `interval_movie_header.html`.

The file, `interval_methods.js` includes many functions for managing a timer, which is a variable stored as a hidden HTML form field in `timing_fields.html`. By recording the playhead position when a user starts to play a movie, and then maintaining a timer, it is possible to determine when the playhead has moved, and the user is no longer continuously watching the video. When a user stops or moves the play head position, this triggers an event that will call `check_and_send()`, which determines if the playhead was moved far enough to conclude that the user has finished watching that video segment. The method will then send an AJAX call to the server using jQuery so that a new ViewInterval object can be created in the database to keep track of users' viewing habits. The event listeners that associate the methods within `interval_methods.js` and corresponding QuickTime events can be found in `interval_movie_header.html`.

4.3.3.2 jQuery

jQuery is a free, widely-used, open source JavaScript library with several features for facilitating client-side HTML scripting. In this project, I used it primarily for putting together client-to-server Ajax calls and overriding the default form submission behavior for the comment and favorite forms. When an HTML form is submitted, this will redirect the user to a reloaded page or a different page. I wanted to be able to allow students to add a comment or create a video as a favorite without needing to refresh the page, because refreshing the page would reset all of the local timing variables for that video page, which would reset the position of the play head and disrupt the tracking of users' viewing intervals.

In order to override the default form submission behavior, I included snippets of jQuery code to handle the submission through a function that would send an Ajax request, and block the page-reloading default submission behavior. As an example, I included the following code at the bottom of the HTML comment submission form file:

```
<script type="text/javascript" src="/site_media/comment_submission.js">
$('#comment_form').submit(
  function() {
    submit_comment();
    return false; } );
</script>
```

This code will run when the HTML `comment_form` is submitted. This calls `submit_comment()` from `comment_submission.js` (shown below), which extracts variables from the form fields and then includes them in an Ajax request that it then sends to the server. When that function finishes, the embedded jQuery script returns “false” in order to block the normal HTML form submission.

```
function submit_comment()
{   var username = $('input[name=username]').val();
    var ta_id = $('input[name=ta_id]').val();
    var text = $('textarea[name=text]').val();
    var permissions = $('select[name=permissions]').val();
    console.log("in submit_comment");
    $.ajax({    type: 'POST',
                url: "/comment_update/",
                data: { username : username, text : text,
                        permissions : permissions, ta_id : ta_id, },
                success: function(response){
                   var new_comment = "<tr><td colsapn=\"2\">"
                       + response.text + "<br/> --"
                       + response.username + " at "
                       + response.time +"</td></tr>";
                    $('.comment_display').append(new_comment); },
                dataType: "json", });
    return false; }
```

This function uses the element identifiers from the HTML form to extract variables like 'username' and then creates a dictionary object containing the relevant information about the comment. This information is sent to the server, and upon receiving a response confirming the comment's creation from the server, the method will create a new HTML formatted comment and append it to the existing `comment_display` table. This provides the illusion of reloading the entire comment query set for a given video, by appending identical HTML code to the end of the existing table, but it doesn't require changing anything except the comment table, so the viewing interval and favorite variables are not affected. This is also especially important because the favorite button is also an HTML form, so a single submission would cause both to be submitted, and this would create a host of other problems with disambiguation as well. With jQuery, it is easy to override the default submission behavior to package up the form submission behavior into an Ajax call to the relevant submission handler on the client side. The server side behavior for handling Ajax requests is discussed in further detail in the next section.

4.3.3.3 Ajax

Django provides some very helpful, built-in functionality for dealing with Ajax requests on the server side. The jQuery functions send Ajax request data in the form of a serializable dictionary. I chose to use JavaScript Object Notation (JSON) because Django provides nice serializers to convert objects into this format, but the choice is otherwise arbitrary. The requests are sent to a particular URL, which, in turn, sends them to the associated view function as specified in `urls.py`. This view function sees an incoming HTML request, and then checks to make sure that it is an Ajax request by using the built-in `is_ajax` Boolean that Django provides for HTTP requests. The view function can then treat the incoming request as a normal HTTP request object and extract the relevant information from the `request.POST` dictionary, and then use these values to create a new database object before providing an Ajax response to the client-side script to update the user's view for the relevant content. The following example shows this server-side Ajax view function that handles the Ajax request for the comment submission. It continues the example that was started in the previous section:

```
def comment_update(request):
    dict = {"username": '', "text": '', "permissions": '', "ta_id": ''}

    if request.is_ajax():
        print "request in comment_update is ajax"
        if request.method =='POST':
            username = request.POST['username']
            text = request.POST['text']
            permissions = request.POST['permissions']
            ta_id = request.POST['ta_id']
            user = User.objects.get(username=username)
            ta = TopicAssignment.objects.get(pk=ta_id)
            comment = Comment()
            comment.clip = ta
            comment.user = user
            comment.text = text
            comment.permissions = permissions
            comment.save()

            print "permissions are %s" %(permissions)

            dict["username"] = str(username)
            dict["ta_id"] = str(ta_id)
            dict["text"] = str(text)
            dict["time"] = str(comment.time)
        return HttpResponse(simplejson.dumps(dict),
                            mimetype="application/javascript")
```

There are also server-side Ajax request-handling functions to create new Favorite objects and ViewInterval objects in the database. When a user adds a video as a favorite, the server-side function will send back information that changes the button text from "Add Favorite" to "Remove Favorite" without reloading anything else on the page.

4.3.4 Apache

Apache is a web server software package used to provide pages and content in response to client requests. While Django provides a basic development server, this is in no way sufficient for serving large media files like the tutorial videos produced for this system. Each Apache server is set up as a separate virtual host.

4.3.4.1 Basic Configuration

In order to set up the existing Django project with the Apache server, I installed the `mod_wsgi` Python adapter for Apache. In order to point the Apache setup to this, I set up two path variables. The `WSGIScriptAlias` in the Apache setup points to the location of the `django.wsgi` configuration file. The `django.wsgi` file points, in turn, to the root folder for the Django project, the correct path for the current python version, and the correct Django settings module for the project. The `WSGIPythonPath` within the Apache setup points to the root directory of all installed python modules within the application.

4.3.4.2 Log Files for Debugging

Throughout development, I used print based debugging. For server side python scripting, these messages went to a terminal. With Apache in place, these messages did not go directly to the terminal. Instead, I overrode the "print" method to save all print comments to an Apache error log. This needed to be changed for all files that were sending print output to the terminal previously, but the result was that more information could be saved at once.

Chapter 5: Results of User Study

In order to test the effectiveness of the tutorial video system, I introduced it as an extra resource for the Spring 2011 semester of the MIT course "6.004: Computation Structures," better known simply as "6.004." I asked other instructors to try the recording system and provide feedback, and I also collected feedback from students in many forms including indirect tracking of viewing habits, comments on the website, informal paper surveys, and personal conversations. In this chapter I describe the results of the user study in terms of this feedback.

5.1 Testing Environment

6.004 was a good choice for testing the system for several reasons. Most importantly, it is the course that I was teaching for the Spring 2011 semester, and had been teaching for the three semesters prior. I envision this video system a supplement to an existing curriculum, and introducing it in a class that I was teaching allowed me to get feedback from students on the tutorial system during office hours, and made it easier for me to produce content relevant to the current semester.

6.004 was also a good choice because it is a fairly large class, with 150 to 200 students in a typical semester, and a sizable teaching staff of 6 to 8 undergraduate "Lab Assistants" (LAs) and 4 to 5 graduate teaching assistants every semester. Given that response rates are typically below 50% for institutionalized and accepted course feedback practices, having a large number of students seemed to be a good way to try to get useful feedback. The sizeable number of course staff members also provided me with a great source of volunteers to try out the recording process and give me feedback about the system from a content-generation perspective.

Another characteristic of the course that made it a prime candidate for a system like this was the significant lab component, and the dedicated lab space. For Spring 2011, there were 8 lab assignments and 4 quizzes. 6.004 has a well-staffed dedicated lab space, where students will come in for required check-off meetings, to get help on assignments, or to ask for help on tutorial problems to study for quizzes. When trying to record one-on-one interactions with students, it is important to have some of those interactions to record, and the 6.004 lab provided a space where those kinds of interactions are plentiful.

In addition to 8 weekly office hours in the lab, I taught two smaller recitation sections that met twice weekly. This meant I had a group of about 40 students with whom I had more regular interaction. These students were the first to try the tutorial system, and they were very helpful in providing informal, in-person feedback before or after teaching. In addition, over half of these students filled out an optional survey with questions about the video system. While monitoring and feedback mechanisms are built into the tutorial video system, informal communications and paper surveys provide different, equally useful information. In Section 5.3, I provide more detailed results of this feedback.

5.2 Instructor Feedback

When designing a system for recording explanations, it is important to ensure that it is easy for someone to begin using the recording hardware and software for the first time. It is also important to design a system that is *likely* to be used. Authors must feel comfortable with the whole process and the benefits to participating in creating new media must outweigh the time investment. In order to test the effectiveness of the recording setup, I collected feedback from five other volunteers.

Overall, not as many authors contributed material as I had hoped, and but the participants did provide some valuable insight regarding the factors that influenced their level of contribution. The authors reported that the recording technology and video upload process were sufficiently intuitive, but that there were larger concerns that hindered their interest in creating tutorial videos. Surprisingly, these barriers were mostly related to the general awkwardness of "explaining" something to a machine instead of a human, and the self-consciousness of being recorded.

Overwhelmingly, the recording volunteers reported feeling nervous by knowing that they were being recorded, but said that if "everyone was doing it" that they would probably not be as nervous. The instructors seemed to be a lot less comfortable with making mistakes when they were being recorded. Most said that this wouldn't go away entirely, but that they would be much more relaxed about the idea if there were a larger set of contributing instructors from past and present semesters. Surprisingly, the instructors needed face-to-face feedback just as much as students did, if not more.

5.2.1 The Volunteers

I and five other volunteers were able to provide usability assessments after learning to use the recording hardware and software. Not all of the recording volunteers published content through the website for the class, but all of the instructors had experience with teaching students in a lab and/or classroom setting.

Two of the instructors were undergraduate staff members for the course, and two were graduate teaching assistants in 6.004 during the semester. Another volunteer has been a graduate TA for 6.004 for several semesters, and yet another had been a TA for two other major undergraduate courses at MIT and was interested in producing content for another course.

5.2.2 Recording Environments

The author's narration provides the primary audio component of the tutorial videos. For all of the recordings produced in the course of the project, this was accomplished with a generic, built-in iMac microphone, though more sophisticated hardware certainly exists. The Cintiq is not especially portable, and so there were two main locations for recording. The first was a private office, and then later, the recording setup was moved to the computer lab area for 6.004

5.2.2.1 Recording in Private

Initially, it seemed like a better solution to keep the hardware setup in a location that was free of background noise for the majority of the day. This initial space was a shared office. Despite initial concerns for background noise, it became clear that this was not the primary barrier to producing recordings. Even if the equipment is in a relatively private and isolated location, there are several factors that make it more challenging to produce recordings.

Two volunteers were not active staff members for 6.004. Both tried creating recordings in private. One had experience with teaching the course in the past and was interested in trying to contribute material, and the other was currently engaged in teaching another class and was interested in producing material to share with that course. Both of these participants showed great enthusiasm, and expressed lack of current motivation or active use of the recording system within their own course as personal barriers to participation. One notable fact is that both of these people have had multiple semesters of experience with teaching classroom sections, and they wanted to share this knowledge by recording it. One problem that they noted was that the lack of human feedback was unnatural and made them feel less satisfied with the potential use of future videos when preparing recordings in advance.

Indeed, it seems easier to create videos with a student present to provide a prompt, and this is an important factor in deploying a system like the one described in this thesis. Many good instructors rely on student feedback like facial expressions and body language in order to offer effective explanations. This seems to be just as important as the feedback that students receive while working through concepts with instructors.

There were some advantages to recording in private. The most noticeable difference was in audio clarity. In the lab, many other students were talking, and when this happened close enough to the recording setup, it was sometimes possible to hear others' loudly enough to make out words. A better microphone or more careful positioning of the recording setup might mitigate this, but certain noises are unavoidable. Many instructors also felt more comfortable with practicing in private because of the decreased pressure and minimal distractions.

5.2.2.2 Recording with Students in the Course Lab

The primary motivation of moving the recording setup to the lab was to capture more realistic interactions with students and teachers. A fortunate consequence of this was that it made it much easier for teachers to create recordings of their ad hoc explanations. This did, however, require teachers to be comfortable with recording their unrehearsed explanations. Even though several of the instructors expressed apprehension with being recorded when they felt unprepared, those who tried recording prepared tutorials in private expressed that they felt the absence of students detracted from the eventual quality of the recordings.

I observed that when students understand something is a recording of an impromptu in-lab explanation, they are much more appreciative of supplementary material than they are concerned about the somewhat rough audio quality or occasional backtracking when instructors realize they need to correct mistakes. After creating many videos in private and in the lab with students, it seems that despite the potential for lower quality or mistakes given lack of preparation, that recording explanations in lab and spending minimal time editing recordings is the best way to provide a healthy amount of material. Regardless of the level of preparation, individual students will always have their own reasons for preferring one explanation to another. It is important to ensure that there are enough recordings from enough different instructors that students can find those that are most helpful to them.

The natural "question and answer" style interaction between students and teachers is also a very important way that students learn. Throughout my teaching experience, there have been many times that other students asked to listen in on an explanation that I had begun offering to a single student. They were not concerned with my level of preparation; they simply had the same question and were eager for a new perspective on the topic to help them understand. This is why the less-prepared, in-lab interactions are valuable as educational media, and this is why I believe that it is a good idea to include a means to record these interactions easily in lab or classroom settings.

There are a few disadvantages that were more surprising. As discussed before, teachers felt as if they were put "on the spot" in lab, but I also discovered that some students were incredibly concerned about having the answer to their question recorded. Student privacy is an important issue that should not be overlooked, and if students are concerned about their voices appearing in the audio track of the recording, it might inhibit their normal pattern of interaction with the teacher. Most students were only concerned with making sure that their dialogue with the instructor didn't end up somewhere public like YouTube, and were satisfied with the practice of sharing content with other students in the class. In fact, once students grew accustomed to the device being in lab, some were more eager to ask me to record solutions instead of answering questions manually because they were interested in reviewing the explanation later and also sharing it with their classmate who were confused about the same topics.

5.3 Student Feedback

I collected student feedback through the comment and favorite mechanisms on the site, through user viewing behavior from view history graphs, and through personal surveys. All of these methods provided different information, and the results are discussed in the following sections.

5.3.1 Personal Interviews

Informal interviews were helpful for gathering information about tutorial video use. Many of the students who used the tutorial video system regularly seemed to be in touch with one another. Several students informed me that they discussed the best videos to watch over instant messaging with one another instead of adding videos as "favorites" in the system. When interviewed, these students seemed very excited about the potential for the system and contributions is could make to future academic "generations," but they said that the favorite system didn't seem as attractive. Two students cited the popular five star rating system used by many online retailers and media providers as a more attractive model because it was more familiar.

One student suggested that the "favorite" feature would be more appealing if the course had a Facebook application where students could "like" videos. Yet another student said that the anonymity of adding a favorite seemed to render it unimportant, but also mentioned concerns about privacy, and said that many students might feel intimidated to share this information with their peers for fear of seeming "dumb." Indeed, a small minority of students insisted that their voices be edited out of help videos because they were concerned about others hearing them ask what they worried might be "stupid questions."

Given somewhat scarce amount of favorite actions and commenting for the videos, tracking users' viewing behavior was key in measuring the effectiveness of individual videos. Overall, the graphs provide a simple, visual record of the number of students continuously watching for each part of the video. This can give instructors valuable feedback in determining which sections of an explanation or lecture turned out to be most interesting to students. The information from these graphs can also give some more surprising forms of feedback as discussed in the following sections.

5.3.2 Surveys

There were two kinds of surveys that provided student feedback for the tutorial video system. I composed optional paper surveys for a subset of the class, and also was able to read course reviews published by a third party group. The informal surveys provided more specific responses because they explicitly asked about the videos, but the more open-ended responses students submitted to the third party system provide interesting information as well.

Just over a month after introducing the tutorial video system, I distributed informal surveys to the approximately 50 students in my two recitation sections. These surveys included questions about their general experiences in the class, and they also listed questions about the video system. About two thirds of these responses indicated that the students had started using the system. Among the responses indicating some experience with the system, most students had watched at least three of about 15 videos that were available at the time, and about half of them had watched over 10 of the videos. Most students had very positive commentary about the supplementary material, and several responses included empathic requests to make more videos. A few responses indicated that students would like to see videos available in multiple formats instead of just QuickTime. I did not develop that additional functionality at the time because of concerns with getting the JavaScript view history tracking working with other embedded movie players. In fact, there was a bug in the view history tracking at the time of this survey. I noticed that the number of students viewing videos and the average number of views per students was much higher than what I would have expected given the view history objects in the database.

A student run organization within the Department of Electrical Engineering and Computer Science at MIT, Eta Kappa Nu (HKN), collects anonymous student feedback and provides this information to instructors at the end of the semester. The text of student commentary is available to only to staff members immediately after the term, and HKN makes general summaries of these reviews available to the MIT student community several months later. The course evaluation did not ask specific questions about the new video content, but several students did leave comments about the video system in these overall evaluations. Five out of 65 responses to this course survey mentioned the videos, and all were highly positive. Students said that they found the videos very useful for quiz preparation, and that they preferred the videos to static handouts for explaining concepts. The view history records also show that students are much more interested in the videos right before a quiz, and at that time, they are particularly the ones that demonstrate solutions to old quiz problems.

5.3.3 View History Graphs

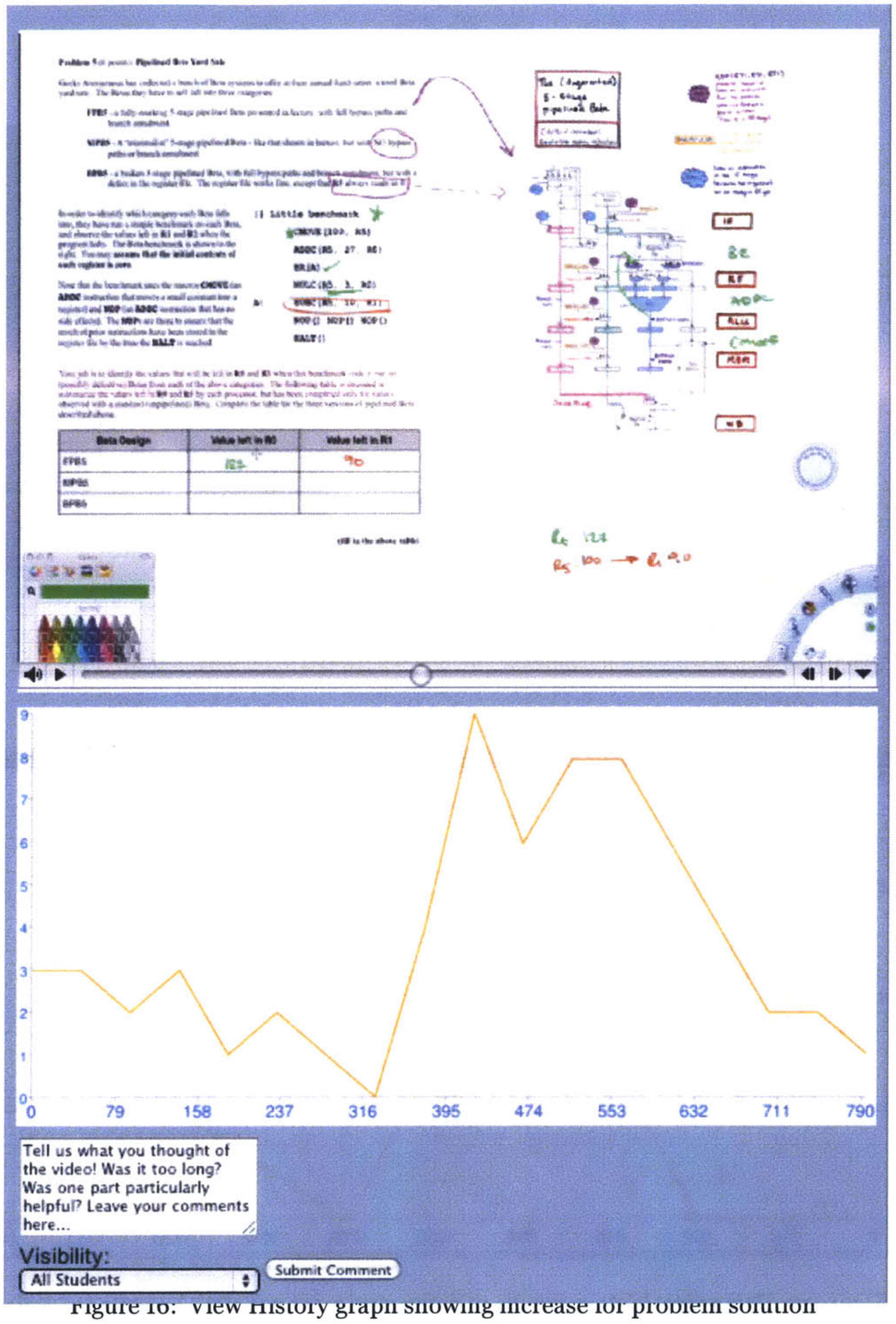
View history graphs provided a highly useful way to measure student interest and participation. Part of the success of these graphs is due to the fact that the tracking occurred automatically, and students did not have to think about comments to generate and share directly. Another reason these graphs were so helpful is that they allow instructors to see *how* a particular video was viewed in order to get more specific feedback. Of course, the graphs do not explicitly provide information about *why* a video was more popular or less particular at a particular point, but observing student viewing trends is enough to draw some very helpful conclusions as discussed in the following sections.

5.3.3.1 Determining Critical Sections

Many of the view history graphs that I observed showed a sharp increase or decrease in the number of users at some point in the video timeline. Trends like this can provide more insight about the kind of information that students were looking for.

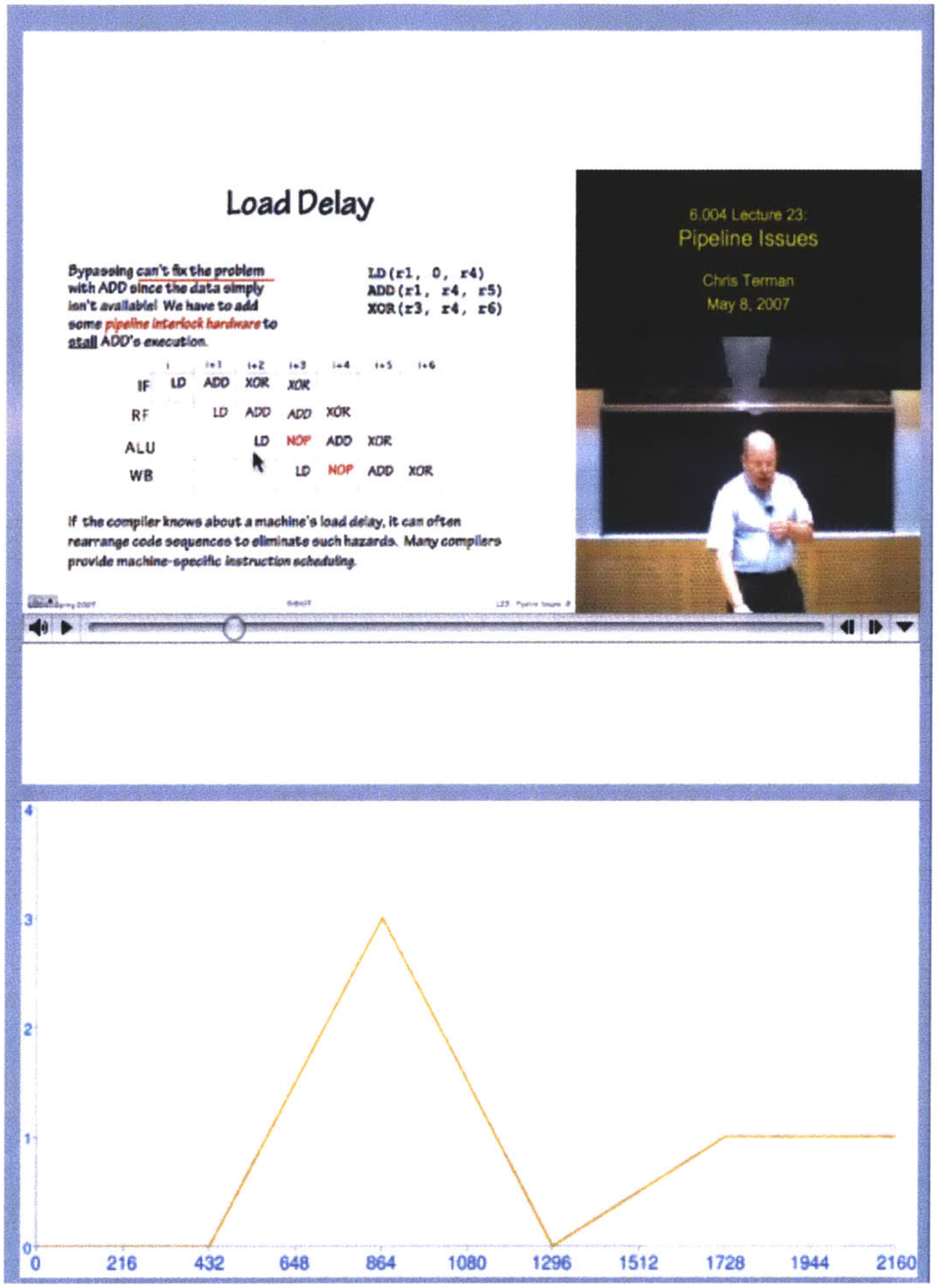
In videos of worked problems, more students seemed to pause and watch the parts of the video where the instructor was simply writing down the answers and then explaining them. Most of the videos began with an instructor reading the problem, interpreting it, and setting it up. The instructor would then work through the example before summarizing key points and answering student questions. An example of the increase in number of viewers during the actual problem-solving time can be seen in Figure 16. The previous part of the video had been more focused on explaining the way to approach problems of this nature and discussing the diagram, which appears in the movie to the right. Then the instructor started to focus on the solution to this specific problem, and more people started watching continuously.

Interestingly, lecture videos often showed less dramatic changes in number of continuous viewers throughout the course of the video, but there were still several key instances of observable trends. This seemed to happen several times when students were viewing information in lecture videos that covered central concepts that were particularly useful for solving typical quiz problems. Figure 17 shows an example of this in a lecture about processor pipelining. The students seemed to start watching the video when they saw the slides about instruction delays and a "waterfall" diagram for depicting pipelined instruction flows. They stopped watching when the instructor began talking about the details of implementing logic behind control signals for the overall behavior.



Tell us what you thought of the video! Was it too long? Was one part particularly helpful? Leave your comments here...

Figure 16: View History graph showing increase for problem solution



Load Delay

Bypassing can't fix the problem with ADD since the data simply isn't available! We have to add some *pipeline interlock hardware* to stall ADD's execution.

```
LD(r1, 0, r4)
ADD(r1, r4, r5)
XOR(r3, r4, r6)
```

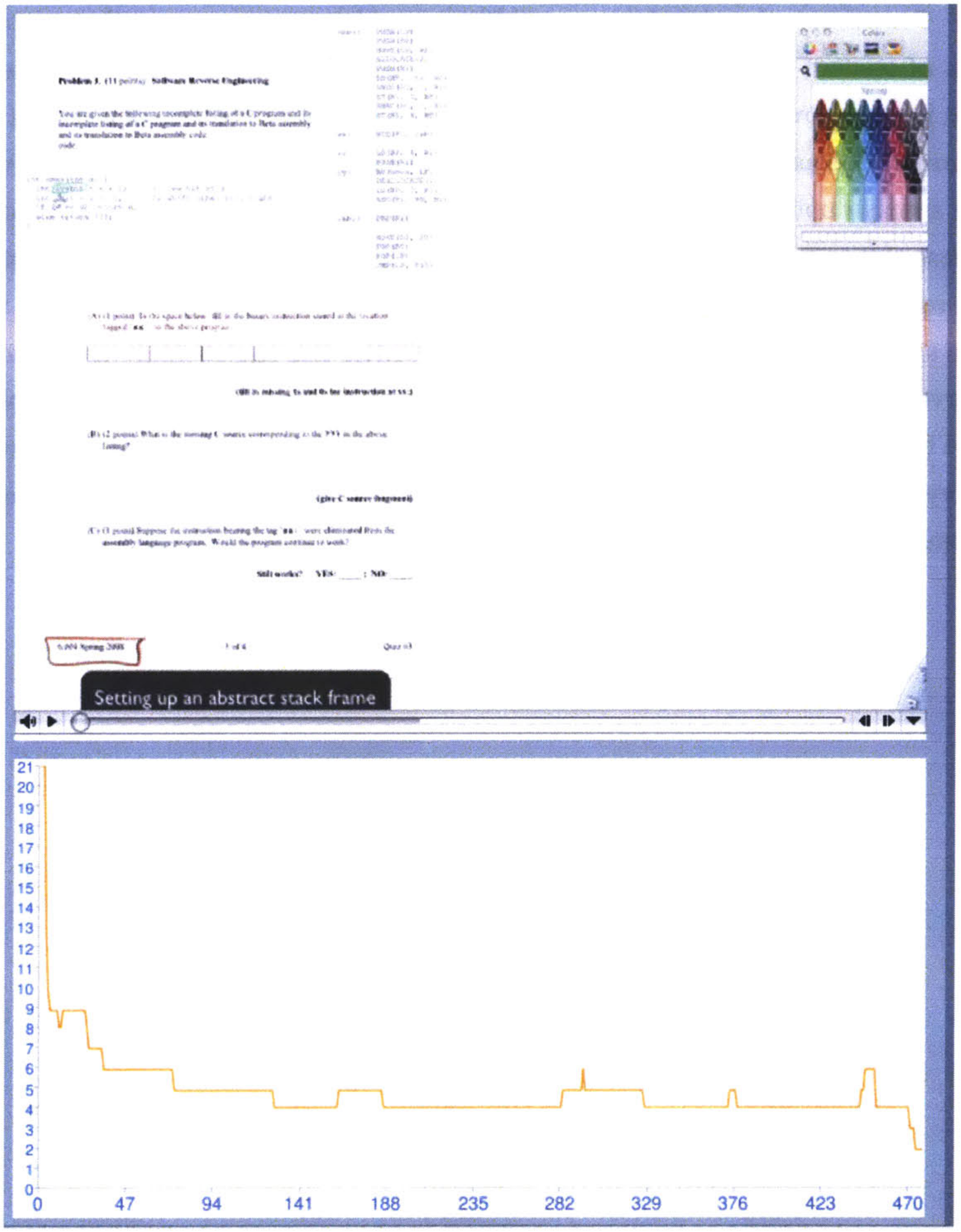
	i	i+1	i+2	i+3	i+4	i+5	i+6
IF	LD	ADD	XOR	XOR			
RF		LD	ADD	ADD	XOR		
ALU			LD	NOP	ADD	XOR	
WB				LD	NOP	ADD	XOR

If the compiler knows about a machine's load delay, it can often rearrange code sequences to eliminate such hazards. Many compilers provide machine-specific instruction scheduling.

Figure 17: Using view history graphs to determine most popular concepts for review

5.3.3.2 Drop off from Poorly Labeled Videos

Sometimes, the view history graphs were able to convey somewhat surprising kinds of student feedback. By observing very unusual trends in particular videos, it made it more obvious when something was amiss with the video. In one case, a video was mislabeled and said to be the continuation of a previous example, when both videos were simply duplicates of the same full worked problem. When students went to view the rest of this example, they noticed that it was the same thing they had just watched. As a result, there was a very sharp drop off within the first few seconds of the movie when students left the page to find new material. After this mislabeling was corrected, I observed the viewing trends starting to even out. Figure 18 is an image of the graph that made this mislabeling obvious in this example, and provides an illustration of how view history graphs can be used to make sure that titles and descriptions are sufficiently informative.



Problem 3. (11 points) Software Reverse Engineering

You are given the following incomplete listing of a C program and its incomplete listing of a C program and its translation to Beta assembly and its translation to Beta assembly code code

(A) (1 point) In the space below, fill in the binary instruction stored at the location tagged 'xx' in the above program.

(fill in missing 1s and 0s for instruction at xx:)

(B) (2 points) What is the missing C source corresponding to the ??? in the above listing?

(give C source fragment)

(C) (1 point) Suppose the instruction bearing the tag 'xx:' were eliminated from the assembly language program. Would the program continue to work?

Still works? YES: ____ ; NO: ____

6.004 Spring 2008 3 of 4 Quiz #3

Setting up an abstract stack frame

5.3.3.3 Comparing Video Types and Student Response

Students seemed to watch entire lectures (with the chapter size set to the length of the video) with less skipping around than they did in watching worked examples. In addition, students seemed to view lectures videos to get a general sense of the material before viewing videos of worked examples. There were, however, fewer viewers overall for the lecture videos, and this could mean that users who did not want to invest a large amount of time in looking for information would go to the shorter worked-example videos first, and then skip around while looking at those videos. An example of a fairly typical lecture-viewing graph is shown in Figure 19.

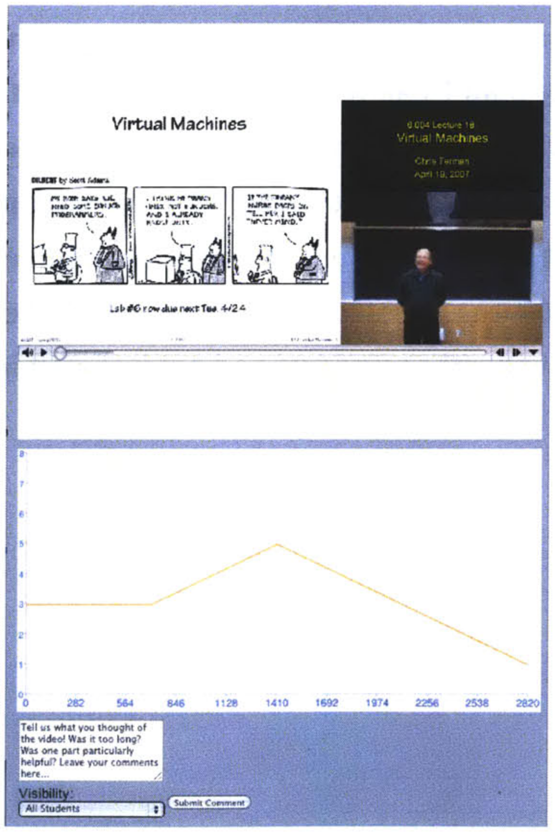

Virtual Machines
Lab #6 now due next Tue 4/24
0 282 564 846 1128 1410 1692 1974 2256 2538 2820
0 1 2 3 4 5 6 7 8
Tell us what you thought of the video! Was it too long? Was one part particularly helpful? Leave your comments here...
Visibility:
All Students
Submit Comment

Figure 19: A More typical view history graph for an hour-long video lecture

5.3.4 Video Comments

Overall, several students provided helpful commentary, but the number of comments was not as high as it could have been. This is, perhaps, the result of students believing that the purpose of commentary was to provide feedback to video authors and nothing more. When polled with optional in-class surveys, as discussed in Section 5.3.2 students had more things to say but these comments were generally positive. The commentary on the website was mostly constructive criticism about how to improve site features or video quality. This indicates that students saw the comment mechanism as a way to communicate with the staff about concerns that they had with the videos rather than a mechanism for discussing video content.

This seems to be a somewhat natural result, though it was a more limited subset of potential commentary than I had hoped for. The comment box default text certainly had some influence on the types of comments that students left. This text said, "Tell us what you thought of the video! Was it too long? Was one part particularly helpful? Leave your comments here.“ It seems that this worked to a certain extent, but a more generic message could have encouraged students to think about the comments as a broader form of communication with other students and not just staff members.

Chapter 6: Future Work

Students in 6.004 and participating instructors seemed to have very positive feedback in general for the system as a whole and the general proof-of-concept trial of the system. Using the system in the classroom demonstrated that there is room for improvement, however, and the following sections discuss some of the significant issues that I observed in the course of this project.

6.1 Rating System

In general, students were less interested in going out of their way to provide feedback through the website directly. Several students sent me email or approached me in person to offer helpful feedback, but the level of indirection necessitated by leaving feedback on the site seemed to make these features less popular. The indirect feedback from the view history graphs provided the most useful information.

The view history tool should not be the only feedback mechanism that produces a large number of data points. It is certainly valuable, but further work in the area should focus on designing more engaging ways for students to provide feedback. As one student suggested, a more social way to share and interact with videos may be helpful. Other students have suggested more sophisticated algorithms for suggesting related videos after users view a particular segment. It is an interesting challenge to provide these more advanced features for the feedback systems without compromising student privacy or the sense of community created by keeping these videos organized for a particular course.

6.2 Chapter Segmentation

The TopicAssignment objects were layered on top of the basic PublicVideo objects in the database. In order to play a TopicAssignment video chapter, JavaScript functions set the playhead to the appropriate position and started and ended the video player at the appropriate times. This approach caused the entire video to be loaded into the page, however, and this was hard to manage with also using playhead information to track user viewing habits to create ViewInterval objects.

Part of the problem with this approach was the added complication and confusion from tracking start and end times for ViewInterval objects relative to the specific TopicAssignment in order to get accurate feedback about viewing habits within that chapter. This created some synchronization bugs because ViewInterval objects could not be accurately recorded until the entire TopicAssignment object has been set up in the page, and this couldn't happen until the entire PublicVideo video file was loaded into the page. If users started watching the video before it was fully loaded, then it was very difficult to accurately collect ViewInterval records.

Another issue with this approach was that students didn't seem to like the fragmented videos as much. This was particularly relevant to the lecture videos. Direct user comments and indirect results from the view history graphs both suggest that students appreciated flexibility when watching lecture videos and did not find the segmentation especially appealing. Creating separate videos instead of building TopicAssignment objects on top of the PublicVideo objects might not mitigate this concern, but it would provide a mechanism for drawing attention to particular clips without introducing the additional confusion about the playback mechanism for students.

6.3 Answering Questions Remotely

The personal nature of the tutorial videos and the fact that they answered specific student questions were both very valuable features. It is not always possible, however, for students to be present for office hours when they have questions.

There are many solutions for answering student questions remotely. For 6.004, the most common method was basic emails. Other courses at MIT and at other universities have started using tools like Piazzza. Piazzza is a system that allows students to post questions and then other students or staff members can respond to these questions such that all other students can view these. This provides a more streamlined way to organize student questions than email, and allows other students to benefit from instructor response to others. There are additional features to allow instructor to view graphs of student activity, and to format their text based responses to students with LaTeX or HTML. This is still incredibly limiting, however, and very far from the natural method of sitting down with a piece of paper and a pencil to explain something to a student.

Integrating part of the tutorial video system with a tool like Piazzza would provide a much more effective means of answering student questions remotely and promoting dynamic records of interactions. This could be done by making a few simple changes to the authentication mechanisms in the tutorial video site to permit teachers to easily direct students to individual explanations, but it would be even more appealing to provide video explanations in line with text comments. It would also be interesting to give students access to one or more machines with hardware for recording explanations to see how they might use the system to create video explanations for one another.

Chapter 7: Summary

In this project, I combined a set of hardware and software to facilitate the production of educational video content. I also created a web-based application to share this content with students and gather their feedback. Once these things were in place, I introduced this system as a supplementary tool in an existing MIT class for a semester. I had other volunteers provide valuable feedback on the recording process, and I analyzed feedback from many students using the system in order to provide recommendations for future systems of this kind.

Overall, the user study demonstrated that students are eager to have new kinds of educational media to help them in their studies, and that they are less willing to provide explicit feedback unless they are having trouble accessing the material. Indirect methods like view history tracking and more personal methods like in-person surveys and discussions provided better feedback. Students appreciated the ability to see the solution to a problem sequentially and many started viewing videos in the middle of an explanation because they were looking for some critical step that they did not quite understand.

This system is a fully functional, standalone tool for educational media recording and distribution, but I also intend it as a helpful starting point for related applications. The results of the user study provide valuable information for those interested in observing the ways that students interact with new forms of educational media, and I expect this information to be relevant to the development of future educational technologies.

Chapter 8: Works Cited

[1] MIT Open CourseWare. (2011). *About OCW.* Retrieved May 2011, from MIT Open Courseware: http://ocw.mit.edu/about/

[2] Khan Academy. (2011). *FAQ.* Retrieved May 2011, from Khan Academy: http://www.khanacademy.org/about/faq

[3] SMART Technologies, Inc. (2010, Oct 19). *SMART Introduces New Class of Interactive Whiteboard.* Retrieved from http://investor.smarttech.com/releasedetail.cfm?ReleaseID=520169

[4] SMART Technolgies, Inc. (2011, Feb 2). *Two New SMART Collaboration Systems Available Globally.* Retrieved from http://smarttech.com/us/About+SMART/About+SMART/Newsroom/Media+releases/English+US/Releases+by+year/2011+media+releases/2011/February+2+2011

[5] Telestream, Inc. (2011). *ScreenFlow Overview and Pricing.* Retrieved May 2011, from http://www.telestream.net/screen-flow/overview.htm

[6] TechSmith, Inc. (2011). *Camtasia Educational Pricing.* Retrieved May 2011, from https://store.techsmith.com/education.asp

[7] TechSmith, Inc. (2011). *TechSmith Products.* Retrieved May 2011, from http://www.techsmith.com/products.asp

[8] Google Inc. (2011). *Description of Data Formats.* Retrieved Feb 2011, from Google Charts API: http://code.google.com/apis/chart/image/docs/data_formats.html

[9] Piazzza Inc. (2011). *Piazzza Documentation.* Retrieved May 2011, from https://www.piazzza.com/manual_v3.pdf

Appendix A: Django Files

A.1 Models

A.1.1 Models.py

```
from django.contrib import databrowse, admin
from django.contrib.auth.models import User, Group
## Using the django admin user model
from django.db import models
from django.conf import settings
from usersite.tutorials.enums import *
import datetime, os

IMAGE_TYPE_LIST=['.jpg', '.gif', '.png', ]
VIDEO_TYPE_LIST=['.mov', '.flv', '.mp4']

def get_upload_location(instance, filename):
    ret = os.path.join('%s' %(instance.author),
                       instance.type,
                       instance.semester,
                       filename)
    return ret


class MediaSubmission(models.Model):
    description = models.CharField(blank=True, max_length=200)
    title = models.CharField(blank=True, max_length=200)
    time = models.DateTimeField(auto_now=False, auto_now_add=True)
    file_name=models.CharField(max_length=64, blank=True)

    class Meta:
        abstract = True


class PublicVideo(MediaSubmission):
    author = models.ForeignKey(User)
```

```
type = models.CharField(max_length=7, choices=VIDEO_CHOICES)
semester= models.CharField(max_length=3, choices=SEMESTER_CHOICES)
file = models.FileField(upload_to = get_upload_location)

def __unicode__(self):
    return u' \"%s\" by %s (%s) ' %(self.file, self.author,
                                   self.semester)
def get_absolute_url(self):
    print "running get_absolute_url for video %s \n" %(self.file)
    return 'http://lecture.csail.mit.edu/site_media/%s'
            %(self.upload_location)

def _get_upload_location(self):
    return get_upload_location(self, self.file_name)
upload_location=property(_get_upload_location)

def _get_filesystem_location(self):
    fs_path = os.path.join(settings.MEDIA_ROOT,
                           self.upload_location)
    print "(_get_filesystem_location) fs_path = %s\n" %(fs_path)
    return fs_path
file_system_location=property(_get_filesystem_location)

def _num_staff_favorites(self):
    favoriter_set = self.userprofile_set.all()
    staff_favorite = 0
    for favoriter_profile in favoriter_set:
        staff_favorite += favoriter_profile.user.is_staff
    return staff_favorite
staff_favorite = property(_num_staff_favorites)

def save(self):
    if self.file:
        if not self.file_name:
            # Checking to make sure this does not already exist
            self.file_name=self.file.name
        upload_location = get_upload_location(self, self.file_name)
        super(PublicVideo, self).save()

def _is_video(self):
    return os.path.splitext(self.file_name)[1] in VIDEO_TYPE_LIST
    is_video = property(_is_video)

    def _is_image(self):
        return os.path.splitext(self.file_name)[1] in IMAGE_TYPE_LIST
    is_image = property(_is_image)

def open_location(filesystem_location, **kwargs):
    # Makes the directory structure in the MEDIA_ROOT directory
    # doesn't work without the following two lines.
    print "(open_location) filesystem_location = %s\n"
           %(filesystem_location)
    if not instance.file_name:
        instance.file_name=instance.file.name
    filesystem_path = "%s%s" %(settings.MEDIA_ROOT,
                               instance.upload_location)
    try: os.makedirs(filesystem_path)
    except: print "(save_video) trying path for %s"
                %(instance.file_name)
    filesystem_location = os.path.join(filesystem_path,
                                       instance.file_name)
    destination = open(filesystem_location, 'wb+')
    print "(save_video) opened location %s\n" %(filesystem_location)
    destination.close()


class TopicAssignment(models.Model):
    video = models.ForeignKey(PublicVideo)
    start_time = models.FloatField(default=0.0)
    stop_time = models.FloatField(default=0.0)
    num_staff_favorites=models.IntegerField(default=0)
    num_student_favorites=models.IntegerField(default=0)
    quiz = models.IntegerField(default=0, max_length=1,
                               choices=QUIZ_CHOICES)
    title = models.CharField(blank=True, max_length=200)
    topic = models.CharField(max_length=128, choices=TOPIC_CHOICES)
    ## constrained to match the choices of topic made available
    ## for study materials on the 6.004 website.

    def __unicode__(self):
        return u'(%d) %s: %s' %(self.id, self.video, self.topic)
```

```
def _get_quiz_string(self):
    return u'%s' %(QUIZ_CHOICES[self.quiz][1])
quiz_verbose = property(_get_quiz_string)

def _get_video_author(self):
    return self.video.author
author=property(_get_video_author)

def _get_video_type(self):
    return self.video.type
type=property(_get_video_type)

def _get_video_semester(self):
    return self.video.semester
semester=property(_get_video_semester)

def set_num_staff_favorites(self, value):
    self.num_staff_favorites=value

def set_num_student_favorites(self, value):
    self.num_student_favorites=value

def inc_num_staff_favorites(self):
    print "increment staff favorites for %s from %d " %(self,
          self.num_staff_favorites)
    self.num_staff_favorites +=1
    self.save()
    print "to %d\n" %(self.num_staff_favorites)

def dec_num_staff_favorites(self):
    print "decrement staff favorites for %s from %d " %(self,
          self.num_staff_favorites)
    self.num_staff_favorites -=1
    self.save()
    print "to %d\n" %(self.num_staff_favorites)

def inc_num_student_favorites(self):
    self.num_student_favorites +=1
    self.save()
    print "to %d\n" %(self.num_student_favorites)
    def dec_num_student_favorites(self):
        self.num_student_favorites -=1
        self.save()
        print "to %d\n" %(self.num_student_favorites)

class ViewInterval(models.Model):
    ta = models.ForeignKey(TopicAssignment)
    user = models.ForeignKey(User)
    start_time = models.FloatField(default=0.0)
    stop_time = models.FloatField(default=0.0)
    time = models.DateTimeField(auto_now=True, auto_now_add=True)

    class Meta:
        unique_together = (('ta', 'user', 'time'))

    def __unicode__(self):
        return u'(Viewer: %s) (Video: %s) (Range: %d - %d)'
               %(self.user, self.ta, self.start_time, self.stop_time)

    def _get_range(self):
        return "( %d, %d )" %(self.start_time, self.stop_time)
    range=property(_get_range)

    def has_second(self, second):
        return ((second<=self.stop_time) and (second>=self.start_time))


class Comment(models.Model):
    COMMENT_PERMISSION_CHOICES = (
        ('students', 'Current and future 6.004 students'),
        ('staff', 'Only 6.004 Staff Members and You'),
        )
    clip = models.ForeignKey(TopicAssignment, related_name='comments')
    user=models.ForeignKey(User)
    text=models.TextField()
    permissions=models.CharField(max_length=64,
                choices=COMMENT_PERMISSION_CHOICES, default='students')
    time = models.DateTimeField(auto_now=True, auto_now_add=True)

    class Meta:
        ordering=['time']
    def __unicode__(self):
        return u'%s: %s' %(self.text, self.clip.topic)


class LinkedWebPage(models.Model):
    name=models.CharField(max_length=50, default="")
    url=models.URLField(default=
                    "http://6004.csail.mit.edu/currentsemester/")
    topic_assignment = models.ForeignKey(TopicAssignment)
    pointer_on_page=models.CharField(max_length=50, default="")



## ------- USER MANAGEMENT ------ ##

class UserProfile(models.Model):
    user = models.ForeignKey(User, unique=True)
    ## User is required to have User.get_profile() work.
    ## It has to be named "user" and refer to a "User" ForeignKey
    ## See Django Book ch 12 for more info
    athena_id = models.CharField(max_length=8, primary_key=True)
    student_id = models.IntegerField(max_length=9, unique=True,
                                null=True, blank=True)

    def __unicode__(self):
        return self.athena_id



class Favorite(models.Model):
    ta = models.ForeignKey(TopicAssignment)
    time = models.DateTimeField(auto_now=True, auto_now_add=True)
    profile = models.ForeignKey(UserProfile)

    class Meta:
        ordering=['time']
        unique_together= (('ta', 'profile'))

    def __unicode__(self):
        return u'\"%s\" favorited by: %s' %(self.ta.video.title,
               self.profile.user.get_full_name())
    def save(self, *args, **kwargs):
        print "overriding favorite object save"
        super(Favorite, self).save(*args, **kwargs)
        if self.profile.user.is_staff:
            self.ta.inc_num_staff_favorites()
        else:
            self.ta.inc_num_student_favorites()

    def delete(self, *args, **kwargs):
        print "overriding favorite object delete"
        super(Favorite, self).delete(*args, **kwargs)
        if self.profile.user.is_staff:
            self.ta.dec_num_staff_favorites()
        else:
            self.ta.dec_num_student_favorites()



## -- SIGNAL UTILS -- ##
## a signal is sent when a User object is being saved.
## Immediately after the User is saved, we want to ensure
## that there is a corresponding UserProfile object created
## with that User as a ForeignKey...
## see "Signals" documentation for more info


def make_profile(sender, instance, **kwargs):
    if (UserProfile.objects.filter(pk=instance.username).count() == 0):
        # If there is not already an instance
        profile=UserProfile(athena_id=instance.username)
        profile.user = instance
        # Can't get this method to save the Student ID so it is
        # IMPORTANT to make sure that the student_id is consistently
        # set in the profile after the User is saved
        profile.save()
    instance.is_active=True

# --- SIGNALS --- ##
models.signals.post_save.connect(make_profile, sender=User)
```

A.2 Views

A.2.1 Views.py

```
from django.shortcuts import render_to_response
from django.http import HttpResponseRedirect, HttpResponse
from tutorials.models import *
from staff_views import get_img_url
from tutorials.filters import TopicAssignmentFilterSet
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import AnonymousUser
from django.template import RequestContext
from django.db.models import Q
from django.conf import settings
from django.utils import simplejson
from django.core import serializers
import datetime, os, re

def get_student_info(request):
    is_authenticated=request.user.is_authenticated()

    if is_authenticated:
        print "user %s is authenticated \n" %(request.user.username)
        student=request.user
        profile=UserProfile.objects.get(user=student)
        return {'student' : student,
                'profile' : profile,
                'is_authenticated': is_authenticated,
                }
    else:
        print "user is NOT authenticated\n"
        return {'is_authenticated':is_authenticated}


def get_student_favorites(request):
    MAX_DISPLAY=5
    student_dict= get_student_info(request)

    if student_dict['is_authenticated']:
        profile = student_dict['profile']
        faves = profile.favorite_set.all()
    else:
        faves = TopicAssignment.objects.none()

    dict = {
        'faves':faves,
        }
    dict.update(student_dict)
    return dict


def render_with_student_context(request, template, dict):
    return render_to_response(template, dict,
                              context_instance=
                              RequestContext(request,
                              processors=[get_student_favorites]))


def show_by_topic(request, topic):
    topic_assignments = TopicAssignment.objects.filter(topic=topic)
    title_string='List of Snippets About %s' %(topic)
    dict = {
        'title_string': title_string,
        'header_string': title_string,
        'ta_query_set': topic_assignments,
        }
    template="topic_assignment_list.html"
    return render_with_student_context(request, template, dict)


def show_by_quiz(request, quiz):
    quiz=int(quiz)
    topic_assignments = TopicAssignment.objects.filter(quiz=quiz)
    title_string='List of Snippets From Quiz %s' %(quiz)
    dict = {
        'title_string': title_string,
        'header_string': title_string,
        'ta_query_set': topic_assignments,
        }
    template="topic_assignment_list.html"
    return render_with_student_context(request, template, dict)


def show_by_author(request, author_username):
    topic_assignments = TopicAssignment.objects.
```

```
                    filter(video__author__username=author_username)
    title_string='List of Snippets by %s' %(author_username)
    dict = {
        'title_string': title_string,
        'header_string': title_string,
        'ta_query_set': topic_assignments,
        }
    template="topic_assignment_list.html"
    return render_with_student_context(request, template, dict)

def show_by_semester(request, semester):
    topic_assignments = TopicAssignment.objects
                    .filter(video__semester=semester)
    title_string='List of Snippets from %s' %(semester)
    dict = {
        'title_string': title_string,
        'header_string': title_string,
        'ta_query_set': topic_assignments,
        }
    template="topic_assignment_list.html"
    return render_with_student_context(request, template, dict)

def show_by_type(request, type):
    topic_assignments = TopicAssignment.objects.filter(video__type=type)
    title_string='List of %s Snippets' %(type)
    dict = {
        'title_string': title_string,
        'header_string': title_string,
        'ta_query_set': topic_assignments,
        }
    template="topic_assignment_list.html"
    return render_with_student_context(request, template, dict)

## deprecated - changed to ajax
def make_comment(request, user, topic_assignment):
    permissions='students'
    if 'permissions' in request.POST.keys():
        permissions = request.POST['permissions']

    text=""
    if 'text' in request.POST.keys():
        text = request.POST['text']
    if not (text == "Write your comment here...."):
        comment=Comment(text=text, permissions=permissions)
        comment.clip=topic_assignment
        comment.user=user
        comment.save()
        print "saved comment %s by %s \n" %(comment.text,
              comment.user.username)


## deprecated - changed to ajax
def add_favorite(request, profile, ta):
    # ADDED LINE TO CONSTRUCT FAVORITE MODEL INSTANCE 9/8/10
    favorite=Favorite()
    favorite.profile=profile
    favorite.ta=ta
    favorite.save()


def rm_favorite(request, profile, ta):
    f = Favorite.objects.filter(ta=ta).filter(profile=profile)
    f.all().delete()


def favorite_post(request):
    dict = { "is_favorite": '', "new_button_text": '' }
    print "in favorite_post"
    if request.is_ajax():
        if request.method=='POST':
            username = request.POST['username']
            ta_id = request.POST['ta_id']
            button_value = request.POST['button_value']
            ta = TopicAssignment.objects.get(pk=ta_id)
            user = User.objects.get(username=username)
            profile = UserProfile.objects.get(user=user)
            button_next = "Error"

            print "button_value " + button_value
            if button_value == "Add Favorite":
                button_next = "Remove Favorite"
                print "isn't a fave and adding"
                ## Making a favorite
                favorite=Favorite()
                print "made a blank favorite"
                favorite.profile=profile
                favorite.ta=ta
                print "saving favorite"
                favorite.save()
            else:
                print "is a fave and deleting"
                f = Favorite.objects.filter(ta=ta).filter(profile=profile)
                f.all().delete()
                button_next = "Add Favorite"
            print "new button text " + button_next
            dict["new_button_text"] = button_next
    return HttpResponse(simplejson.dumps(dict),
           mimetype="application/javascript")



# main page for displaying a single movie with comment and favorite options
def show_media(request, ta_id):
    ta_id = int(ta_id)
    ta = TopicAssignment.objects.get(pk=ta_id)
    linked_problems=LinkedWebPage.objects.filter(topic_assignment__id=ta_id)
    print "got ta with topic = %s\n" %(ta.topic)
    print "got ta with src = %s\n" %(ta.video.get_absolute_url())
    template="show_media.html"

    is_user_favorite=False
    student = AnonymousUser()
    student_info = get_student_info(request)
    # get the profile and authentication info

    comments=Comment.objects.none()
    print "there are %d comments  before filtering \n" %(comments.count())

    # empty query set
    if student_info['is_authenticated']:
        student=student_info['student']
        student_faves = get_student_favorites(request)

        comments = ta.comments.filter(
            Q(permissions='students')
            | ( Q(permissions='staff') & Q(user=student) ) )
        if student.is_staff:
            comments=ta.comments.all()

        if student_faves['faves'].filter(pk=ta_id).count():
            is_user_favorite=True
        if request.method=='POST':
            if 'submit_comment' in request.POST.keys():
                make_comment(request, student, ta)
            if 'add_favorite' in request.POST.keys():
                add_favorite(request, student_info['profile'], ta)
            if 'rm_favorite' in request.POST.keys():
                rm_favorite(request, student_info['profile'], ta)

            for key in request.POST.keys():
                print "requst.POST[%s] = %s \n" %(key, request.POST[key])


    print "there are %d comments \n" %(comments.count())
    print "user.is_authenticated = %s \n" %(request.user.is_authenticated())

    topic_number=TOPIC_NUMBERS[ta.topic]
    dict = {
        'debug': True,
        'selected_ta':ta,
        'topic':ta.topic,
        'linked_problems':linked_problems,
        'is_user_favorite':is_user_favorite,
        'user': student,
        'comments': comments,
        'permissions': ['staff','student'],
        }

    if student.is_staff:
        redirect = "/view_history/" + str(ta_id) + "/"
        return HttpResponseRedirect(redirect)
    return render_with_student_context(request, template, dict)


def landing(request):
    template="topic_list.html"
    dict={}
    return render_with_student_context(request, template, dict)


def tutorial_main(request):
    template = "tutprobs.htm"
    dict = {}
    return render_to_response(template, dict
def tutorial_by_topic(request, topic):
    print "in views.tutorial_by_topic"
    template="%s" %(topic)
    print "topic is %s \n" %(topic)
    dict = {}
    return render_to_response(template, dict)


## shows the tutorial problems for the topic assigned to this clip
## want to refine to break this up by problem.
def tutorial_by_id(request, topic, linked_problem_id):
    print "\n in tutorial_by_id\n"
    base_url = "http://6004.csail.mit.edu/currentsemester/tutprobs/"
    page = TUTORIAL_PROBLEM_URLS[topic]
    lp_id = int(linked_problem_id)
    lp = LinkedWebPage.objects.get(pk=lp_id)
    lp_pointer = lp.pointer_on_page
    direct_to = base_url + page + lp_pointer
    print "redirecting from tutorial by id for id=%d to %s\n"
          %(lp_id, direct_to)
    return HttpResponseRedirect(direct_to)




## server side handler function for view intervals
def post_interval_handler(request):
    print "In post_handler \n"

    dict = { "img_div" : '',
             "x_axis_max" : '' }

    message = 'failure'
    interval = ViewInterval()
    if request.is_ajax():
        print "request in post handler is ajax \n"
        if request.method == 'POST':

            iform_start = request.POST['iform_start']
            print "iform_start : %s" %(iform_start)
            iform_end = request.POST['iform_end']
            print "iform_end : %s" %(iform_end)
        username = request.POST['user']
        user = User.objects.get(username=username)
        print "user : %s" %(user)
        ta_id = int(request.POST['ta_id'])
        print "ta_id : %s" %(ta_id)
        ta = TopicAssignment.objects.get(pk=ta_id)
        print "ta : %s" %(ta)
        interval = ViewInterval()
        print "made an interval"
        interval.ta = ta
        interval.user=user
        interval.start_time = iform_start
        interval.stop_time = iform_end
        interval.save()
        # make the interval and the graph to send back
        # but only if it came from the staff view page

        if 'url_match' in request.POST.keys():
            print "url_matched as %s" %(request.POST['url_match'])

            x_length =  request.POST['ta_length']
            print "x_length = %s" %(x_length)
            img_url = get_img_url(ta_id, x_length)
            img_div = "<div id=\"view_graph_div\" class=\"view_graph_div\"
                      style=\"float:left;align:left\">"
            img_div = img_div + "<img id=\"view_graph\"
                      name=\"view_graph\" src=\"" + img_url + "\" />"
            img_div = img_div + "</div>"
            message = img_div
            dict["img_div"] = img_div
            dict["x_axis_max"] = request.POST['ta_length']
            #print "dict[img_div] : %s" %(dict["img_div"])
            #print "dict[x_axis_max] : %s" %(dict["x_axis_max"])
            ##interval = ViewInterval(ta=ta, user=user,
                         start_time=iform_start, end_time=iform_end)
        else :
            dict = {}

print "saved interval"
return HttpResponse(simplejson.dumps(dict),
       mimetype="application/javascript")
```

```
def post_test(request):
    template = "js_test.html"
    ta_id = 4
    ta = TopicAssignment.objects.get(pk=ta_id)
    linked_problems=LinkedWebPage.objects.filter(topic_assignment__id=ta_id)
    is_user_favorite=False
    student = AnonymousUser()
    student_info = get_student_info(request)
    # get the profile and authentication info
    comments=Comment.objects.none()

   # empty query set
    if student_info['is_authenticated']:
        student=student_info['student']
        student_faves = get_student_favorites(request)
        comments = ta.comments.filter( Q(permissions='students')
                                  | ( Q(permissions='staff') &
                                  Q(user=student) ) )
        if student.is_staff:
            comments=ta.comments.all()
        if student_faves['faves'].filter(pk=ta_id).count():
            is_user_favorite=True
        if request.method=='POST':
            if 'submit_comment' in request.POST.keys():
                make_comment(request, student, ta)
            if 'add_favorite' in request.POST.keys():
                add_favorite(request, student_info['profile'], ta)
            if 'rm_favorite' in request.POST.keys():
                rm_favorite(request, student_info['profile'], ta)
            for key in request.POST.keys():
                print "requst.POST[%s] = %s \n" %(key, request.POST[key])
    topic_number=TOPIC_NUMBERS[ta.topic]
    dict = {
        'debug': True,
        'selected_ta':ta,
        'topic':ta.topic,
        'linked_problems':linked_problems,
        'is_user_favorite':is_user_favorite,
        'user': student,
        'comments': comments,
        'permissions': ['staff','student'],
        }
    return render_with_student_context(request, template, dict)
```

```
def comment_update(request):
    print "updating comment"

    dict = {"username": '', "text": '', "permissions": '', "ta_id": ''}

    if request.is_ajax():
        print "request in comment_update is ajax"
        if request.method =='POST':
            username = request.POST['username']
            text = request.POST['text']
            permissions = request.POST['permissions']
            ta_id = request.POST['ta_id']
            user = User.objects.get(username=username)
            ta = TopicAssignment.objects.get(pk=ta_id)
            comment = Comment()
            comment.clip = ta
            comment.user = user
            comment.text = text
            comment.permissions = permissions
            comment.save()

            print "permissions are %s" %(permissions)

            dict["username"] = str(username)
            dict["ta_id"] = str(ta_id)
            dict["text"] = str(text)
            dict["time"] = str(comment.time)
        return HttpResponse(simplejson.dumps(dict),
               mimetype="application/javascript")
```

A.2.2 Student_views.py

```
import sys
sys.stdout = sys.stderr
from django.shortcuts import render_to_response
from django.http import HttpResponseRedirect
#from records.models import *
from tutorials.models import *

from django.contrib.auth.models import AnonymousUser
from tutorials.filters import TopicAssignmentFilterSet
from django.contrib.auth.decorators import login_required

from django.template import RequestContext

from django.conf import settings
import datetime, os, re
import utils



def get_student_info(request):
    is_authenticated=request.user.is_authenticated()

    if is_authenticated:
        print "user %s is authenticated \n" %(request.user.username)
        student=request.user
        profile=UserProfile.objects.get(user=student)
        return {'student' : student,
                'profile' : profile,
                'is_authenticated': is_authenticated,
                }
    else:
        print "user is NOT authenticated\n"
        return {'is_authenticated':is_authenticated}


def get_public_videos(request):
    MAX_DISPLAY=5
    quizzes = TopicAssignment.objects.filter(video__type='OldQuiz')
    labs = TopicAssignment.objects.filter(video__type='LabHint')
    concepts = TopicAssignment.objects.filter(video__type='Concept')
    tutprobs = TopicAssignment.objects.filter(video__type='TutProb')
    all_vids = TopicAssignment.objects.all()
```

```
# located in views.py

    dict = {
        'quizzes': quizzes,
        'labs':labs,
        'concepts': concepts,
        'tutprobs': tutprobs,
        'all_vids':all_vids,
        }
    return dict

def favorites_to_tas(favorite_set):
    return (favorite.ta for favorite in favorite_set)


def get_student_favorites(request):
    MAX_DISPLAY=5
    student_dict= get_student_info(request)
    profile = student_dict['profile']
    fav_quizzes = profile.favorite_set.filter(ta__video__type='OldQuiz')
    fav_labs = profile.favorite_set.filter(ta__video__type='LabHint')
    fav_concepts = profile.favorite_set.filter(ta__video__type='Concept')
    fav_tutprobs = profile.favorite_set.filter(ta__video__type='TutProb')
    faves = profile.favorite_set.all()
    dict = {
        'fav_quizzes': fav_quizzes,
        'fav_labs': fav_labs,
        'fav_concepts': fav_concepts,
        'fav_tutprobs': fav_tutprobs,
        'faves':faves,
        }
    dict.update(student_dict)
    return dict

def get_student_favorite_tas(request):
    MAX_DISPLAY=5
    student_dict= get_student_info(request)
    profile = student_dict['profile']
    fav_quizzes = profile.favorite_set.filter(ta__video__type='OldQuiz')
    fav_labs = profile.favorite_set.filter(ta__video__type='LabHint')
    fav_concepts = profile.favorite_set.filter(ta__video__type='Concept')
    fav_tutprobs = profile.favorite_set.filter(ta__video__type='TutProb')
    faves = profile.favorite_set.all()
    dict = {
        'fav_quizzes': fav_quizzes,
        'fav_labs': fav_labs,
        'fav_concepts': fav_concepts,
        'fav_tutprobs': fav_tutprobs,
        'faves':faves,
        }
    dict.update(student_dict)
    return dict

def get_faves_by_topic(request):
    student_dict=get_student_info(request)
    student_faves = student_dict['profile'].favorite_set.all()
    topic_choices = [ [topic[0], topic[1]] for topic in TOPIC_CHOICES]
    faves_by_topic={}
    for topic_tuple in topic_choices:
        topic_faves = student_faves.filter(ta__topic=topic_tuple[0])
        ## get all TopicAssigned video clips that match the exact topic part
        entry=''
        for topic_fave in topic_faves:
            entry = '%s <td><a href=\"%s\">%s</a></td>' %(entry,
                    topic_fave.ta.video.get_absolute_url(),
                    topic_fave.ta.video.file_name)
            faves_by_topic[topic_tuple[1]]=entry
    return faves_by_topic


def get_fave_tas_by_topic(request):
    student_dict=get_student_info(request)
    student_faves = student_dict['profile'].favorite_set.all()
    topic_choices = [ [topic[0], topic[1]] for topic in TOPIC_CHOICES]
    faves_by_topic={}
    for topic_tuple in topic_choices:
        topic_faves = student_faves.filter(ta__topic=topic_tuple[0])
        ## get all TopicAssigned video clips that match the exact topic part
        entry=''
        for topic_fave in topic_faves:
            entry = '%s <td><a href=\"%s\">%s</a></td>' %(entry,
                     topic_fave.ta.video.get_absolute_url(),
                     topic_fave.ta.video.file_name)
            faves_by_topic[topic_tuple[1]]=entry
        return faves_by_topic
```

```
@login_required
def preview_and_set_topic(request, video_id):

    for item in request.POST.keys():
        print "request.POST[%s] = %s\n" %(item, request.POST[item])

    if not request.user:
        html = "<h2> Error: you are not <a href=\"accounts/login.html\">
               logged in.</a></h2>"
        return render_to_response(html, {})
    if not request.user.is_staff:
        html = "<h2> Error: you are not <a href=\"accounts/login.html\">
               logged in as staff.</a></h2>"
        return render_to_response(html, {})
    video_id=int(video_id)
    video = PublicVideo.objects.get(pk=video_id)
    dict = { 'video':video }
    template="movie_preview.html"
    return render_to_response(template, dict)



## Main view for media browser
## use built in decorator to limit access to logged in users
@login_required
def student_portal(request, topic_snippet_id="", show='All', query_string=''):
    all_topic_assignments = TopicAssignment.objects.all()
    if topic_snippet_id =="":
        #topic_snippet_id=TopicAssignment.objects.all()[0].id
        topic_snippet_id = utils.get_random_ta_id()

    if not request.user:
        return HttpResponseRedirect("/public/")
    if not request.user.is_authenticated():
        return HttpResponseRedirect("/public/")
    #if request.user.is_staff:
    #    return HttpResponseRedirect("/view_history/")
    public_ta_dict = get_public_videos(request)
    favorite_dict = get_student_favorites(request)
    fave_tas_by_topic = get_fave_tas_by_topic(request)
    ta_id = int(topic_snippet_id)
    all_topic_assignments=public_ta_dict['all_vids']
    selected_ta = all_topic_assignments.get(pk=ta_id)
    if 'QUERY_STRING' in request.META.keys():
        query_string='?'+request.META['QUERY_STRING']
    query=query_string

    is_favorite=u'False'
    if favorite_dict['faves'].filter(ta__pk=ta_id):
        is_favorite=u'True'
    verbose_topics = [ topic[1] for topic in TOPIC_CHOICES ]
    pre_filter = TopicAssignment.objects.all()
    if show=='Favorites':
        inner_queryset = favorite_dict['faves'].values('ta')
        pre_filter = TopicAssignment.objects.filter(id__in=inner_queryset)

    # only show a student's favorites
    filterset=TopicAssignmentFilterSet(request.GET, queryset=pre_filter)

    dict={
        'query_string':query,
        'topic_assignment_filterset':filterset,
        'all_topic_assignments':pre_filter,
        'selected_ta':selected_ta,
        'verbose_topics':verbose_topics,
        'faves_by_topic':fave_tas_by_topic,
        'show': show,
        }

    print "(student_portal) size of pre_filter = %d" %(pre_filter.count())

    for item in pre_filter:
        print "(student_portal) pre_filter(n): %s\n" %(item)

return render_to_response(template, dict,
                          context_instance =
                          RequestContext(request,
                                         processors=[get_student_favorites]))

## public browser. no authentication or favoriting.
def browse(request, topic_snippet_id='', is_favorite=False, query_string=''):
    all_topic_assignments = TopicAssignment.objects.all()
    if topic_snippet_id =='':
        topic_snippet_id='1'
        ids = [ta.id for ta in all_topic_assignments]
    ta_id = int(topic_snippet_id)
    print >> sys.stderr, "ta_id = %d\n" %(ta_id)
    selected_ta = all_topic_assignments.get(pk=ta_id)
    selected_video = selected_ta.video
    v_id = selected_video.id

    all_videos=PublicVideo.objects.all()
    selected_video = all_videos.get(pk=v_id)
    print "vid is %d" %(v_id)

    student = AnonymousUser()
    if request.user.is_authenticated():
        student = request.user

    if 'QUERY_STRING' in request.META.keys():
        if not request.META['QUERY_STRING'] =='':
            query_string='?'+request.META['QUERY_STRING']
    query=query_string
    print "full_path = %s\n" %(query)
    filterset=TopicAssignmentFilterSet(request.GET,
                                       queryset=TopicAssignment.objects.all())
    dict={
        'query_string':query,
        'all_videos':all_videos,
        'all_topic_assignments':filterset,
        'selected_ta':selected_ta,
        'user': student,
        }

    template="browse.html"
    response = render_to_response(template, dict)
    return response


def media_browser(request):
    all_videos = PublicVideo.objects.all()
    v_id=3
    # for now, "Timothy Leary's Calenday App - one of the smaller ones.
    print "vid is %d" %(v_id)
    all_topic_assignments = TopicAssignment.objects.all()

    staff_faves={}
    all_faves={}

    # -- stuff to keep the right video-topic thing selected when
    # -- the filter list is modified
    query_string=request.META['QUERY_STRING']
    print "query_string = %s\n" %(query_string)
    v_id_field = 'v_id' in request.GET.keys()
    start_char=u'?'
    full_path=''
    if v_id_field:
        v_id=request.GET['v_id']
    full_path=re.sub("\?*v_id=[^&]*", '', request.get_full_path())
    if 'topic' in request.GET.keys():
        full_path=re.sub("\&*v_id=[^&]*", '', request.get_full_path())
        start_char=u'&'
    # -- end query string processing

    selected_video = all_videos.get(pk=v_id)
    print "full_path = %s\n" %(full_path)

    for k in request.GET.keys():
        print "request.GET[%s] = %s\n" %(k, request.GET[k])
    for video in all_videos:
        favoriter_set = video.userprofile_set.all()

staff_faves[video.id]=favoriter_set.filter(user__is_staff=True).count()
                     all_faves[video.id]=favoriter_set.count()
                     filterset=TopicAssignmentFilterSet(request.GET,
                     queryset=TopicAssignment.objects.all())

    dict={
        'full_path':full_path,
        'start_char': start_char,
        'all_topic_assignments':filterset,
        'selected_video':selected_video,
        'staff_faves':staff_faves,
        'all_faves':all_faves,
        }

    template="browse.html"
    response = render_to_response(template, dict)
    print "the outgoing response is %s\n" %(response.content)
    return response
```

A.2.3 Staff_views.py

```
from tutorials import loader
from django.shortcuts import render_to_response
from django.http import HttpResponseRedirect
from tutorials.models import *
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import AnonymousUser
from tutorials.forms import PublicVideoForm



def check_staff(request):
    if not request.user:
        return HttpResponseRedirect('accounts/login/')
    if not request.user.is_staff:
        return HttpResponseRedirect('accounts/login/')


@login_required
def preview_and_set_topic(request, video_id):
    print "in preview_and_set_topic\n"
    #for item in request.POST.keys():
    #    print "request.POST[%s] = %s\n" %(item, request.POST[item])
    #check_staff(request)
    topic_choices = [ [topic[0], topic[1]] for topic in TOPIC_CHOICES]
    quiz_choices = [ [quiz[0], quiz[1]] for quiz in QUIZ_CHOICES]
    print "about to get video # %s \n" %(video_id)
    video_id=int(video_id)
    video = PublicVideo.objects.get(pk=video_id)

    end_time_units=0;
    start_time_units=0;
    quiz="0"
    topic=""
    title=""

    if request.is_ajax():
        if request.method == 'POST':
            start_time_units = int(request.POST['start_time_units'])
            end_time_units = int(request.POST['end_time_units'])
            topic = request.POST['topic']
            quiz = request.POST['quiz']
    else:
```

```
    if request.method=="POST":
        # We have an upload submission from movie_preview.html
        print "post!"
        if 'start_time_units' in request.POST.keys():
            # then the button, with input type "onclick"
            # triggers the event listener
            # in movie_preview.html "set_start_time();"
            # and here, this the dictionary below,
            # we have the conversion from start_time_units
            # to start_time in the
            # topic_assignment object itself.
            start_time_units = int(request.POST['start_time_units'])
        else: start_time_units = 0
        print "start_time_units = %s \n" %(start_time_units)

        if 'end_time_units' in request.POST.keys():
            end_time_units = int(request.POST['end_time_units'])
        else: end_time_units = 0
        if 'topic' in request.POST.keys():
            topic = request.POST['topic_name']
        if 'title' in request.POST.keys():
            title = request.POST['title']
        if 'quiz_name' in request.POST.keys():
            quiz = request.POST['quiz_name']
        else: quiz="0"

        print "end_time_units = %d \n" %(end_time_units)
        #        print "quiz is %s \n" %(quiz)
        print "topic is %s\n" %(topic)
        print "title is %s \n" %(title)
        ta = TopicAssignment(video = video,
                             start_time = start_time_units,
                             stop_time = end_time_units,
                             topic = topic,
                             quiz = quiz,
                             title = title)
        ta.save()
        print "saved the TA with id : %d \n" %(ta.id)
        return HttpResponseRedirect('/topic_assign/%d/' %(video_id))

dict = { 'video': video, 'topic_choices': topic_choices,
         'quiz_choices': quiz_choices, }

template="movie_preview.html"
```

```
    return render_to_response(template, dict)


@login_required
def selected_video_for_assignment(request):
    check_staff(request)
    selected_video_id = "1"
    if request.method == 'POST':
        print "got a post \n"
        if 'selected_video' in request.POST.keys():
            selected_video_id = request.POST['selected_video']

        else:
            return HttpResponseRedirect("/upload_video/")
        vid=int(selected_video_id)
        topic_choices = [ [topic[0], topic[1]] for topic in TOPIC_CHOICES]
        quiz_choices = [ [quiz[0], quiz[1]] for quiz in QUIZ_CHOICES]
        video = PublicVideo.objects.get(pk=vid)

        dict = {
            'video': video,
            'topic_choices': topic_choices,
            'quiz_choices': quiz_choices,
            }

        template="movie_preview.html"
        return render_to_response(template, dict)



@login_required
def select_video_for_assignment(request):
    check_staff(request)
    print "in select_video_for_assignment \n"
    if request.method=="POST":
        v_id = request.POST['selected_video']
        print "selected video # %s chosen for topic assignment \n" %(v_id)
        vid = int(v_id)
        video = PublicVideo.objects.get(pk=vid)
    videos = PublicVideo.objects.all()
    dict = { 'videos': videos }
    template = "select_video_for_assignment.html"
    return render_to_response(template, dict)
```

```
@login_required
def upload_video(request):

    # makes sure that the user is staff before rendering the page
    check_staff(request)

    if request.method=="POST":
        video_form = PublicVideoForm(request.POST, request.FILES)
        if not video_form.is_valid():
            template = "<h2> Please check the form submission
                        and try again </h2>"
            return render_to_response(template, {})
        else:
            video = video_form.save(commit=False)
            file = request.FILES['file']
            video.file=file
            print "video.file.name = %s" %(video.file.name)
            print "file name is %s" %(file.name)
            video.save()
            return HttpResponseRedirect('/topic_assign/%d/' %(video.id) )
    else:
        video_form = PublicVideoForm()
    videos = PublicVideo.objects.all()
    dict = {
        'videos': videos,
        'user':request.user,
        'video_form':video_form,
        }
    template = 'upload_video.html'
    return render_to_response(template, dict)


def display_interval_list(request):
    topic_assignments = TopicAssignment.objects.all()
    title_string = "List of Topic-Assigned Clips"
    header_string = "Click one of the links to check the view history"
    dict = {
        'title_string' : title_string,
        'header_string' : header_string,
        'ta_query_set' : topic_assignments
        }
    template = "view_history_list.html"
    return render_to_response(template, dict)

def get_img_url(ta_id, x_length="0"):
    print "(get_img_url)"
    id=int(ta_id)
    ta = TopicAssignment.objects.get(pk=id)
    #print "(display_interval_views) ta = %s" %(ta)
    intervals = ta.viewinterval_set.all()
    number = intervals.count()
    interval_stop_times = intervals.values_list('stop_time',
                          flat=True).order_by('-stop_time')
    if not (interval_stop_times.count() == 0):
        intervals_max = int(interval_stop_times[0])
    else:
        intervals_max = 100

    x_length = int(x_length)
    if (x_length == 0):
        x_length = intervals_max

    #print "x_length = " + str(x_length)
    #print "intervals_max = " + str(intervals_max)
    #print "(display_interval_views) there are %d
            intervals for this topic assignment." %(number)
    view_vector = [0]
    step = int(intervals_max/100)
    for i in range(0, intervals_max, step):
        #print "in outer for loop with index = " + str(i)
        view_vector.append(0)
        # initialize array value for this index to zero
        for interval in intervals:
            if interval.has_second(i):
                #print "view_vector["+str(i)+"] = " + str(view_vector[i])
                view_vector[int(i/step)] = view_vector[int(i/step)] + 1

    #for i in range (intervals_max):
    #    print "-- view_vector["+str(i)+"] = " + str(view_vector[i])
    max_views = max(view_vector)
    if (max_views == 0):
        max_views = 1

    #print "max_views = " + str(max_views)
    img_url = "http://chart.googleapis.com/chart?"
    # to make a line chart
    img_url = img_url + "cht=lc&"
    # make a graph that is 800x300 (default)
    img_url = img_url + "&chs=600x300"
    # add the data values
    img_url = img_url + "&chd=t:" + str(view_vector[0]*100/max_views)
    view_vector_length=len(view_vector)
    for i in range (0, view_vector_length, step):
        img_url = img_url + ","+ str(view_vector[i]*100/max_views)

    # format the axis scale and color, respectively
    img_url = img_url + "&chxt=x,y&chxr=0,0," + str(x_length) +
              "," + str(x_length/10) +"|1,0,"+str(max_views)+",1"
    img_url = img_url + "&chxs=0,2244FF,12,0,lt|1,0055FF,10,1,lt"

    return img_url


# want to count all of the people viewing at each time unit to determine
# popular parts of a video. this one just counts all views, without regard to
user
@login_required
def display_interval_views(request, ta_id):
    user = AnonymousUser()
    if request.user.is_authenticated():
        user = request.user
    id=int(ta_id)
    ta = TopicAssignment.objects.get(pk=id)

    img_url = get_img_url(ta_id)
    print "img_url = " + img_url
    context = {
        "ta_start" : ta.start_time,
        "ta_id":ta_id,
        "user":user,
        "ta_stop" : ta.stop_time,
        "selected_ta" : ta,
        "img_url" : img_url
        }
    template = "display_interval_views.html"
    return render_to_response(template, context)
```

A.3 Project Configuration

A.3.1 Urls.py

```
from django.conf.urls.defaults import *
from django.views.generic import list_detail
from django.contrib import databrowse
from django.contrib.auth.views import login, logout, logout_then_login,
                                      password_change, password_change_done
from usersite import views, student_views, staff_views
#enable the admin
from django.contrib import admin
from django.contrib.auth.models import User, Group

admin.autodiscover()

student_list_info = {
   'queryset': User.objects.filter(is_staff=False),
   'template_name': 'student_list.html'}

student_detail_info = {
    'queryset': User.objects.filter(is_staff=False),
    'template_object_name': 'student' }


urlpatterns = patterns('',
                       ## server side interval handler
                       (r'^post_interval/$', views.post_interval_handler),
                       (r'^post_test/page/$', views.post_test),

                       # admin portal
                       (r'^admin/', include(admin.site.urls)),

                       (r'^accounts/login/$', login),

                       (r'^accounts/logout/$', logout_then_login),

                       (r'^accounts/changepw/$', password_change),

                       (r'^accounts/changepwdone/$', password_change_done),
# template: /templates/tutorials/browse.html
(r'^accounts/profile/$', student_views.student_portal),


# handler for uploading comments
(r'^comment_update/$', views.comment_update),

     # handler for adding a favorite
(r'^add_favorite/$', views.favorite_post),


# staff editing and topic assigning page
(r'^topic_assign/(?P<video_id>\d+)/$',
 staff_views.preview_and_set_topic),

# staff video upload
(r'^upload_video/$', staff_views.upload_video),

(r'^select_video_for_assignment/$',
 staff_views.select_video_for_assignment),

# Number of views by movie timeline
(r'^view_history/$',
 staff_views.display_interval_list),

(r'^view_history/(?P<ta_id>\d+)/$',
staff_views.display_interval_views),

# Used to test variable landing pages to bring
# attention to new features
(r'^landing/$', views.landing),

# Select an existing video to add a new chapter
(r'^selected_video_for_assignment/$',
staff_views.selected_video_for_assignment),

# show lists of videos with these attributes
(r'^web/show_media/(?P<ta_id>\d+)/$',
views.show_media),

(r'^web/show_for_topic/(?P<topic>\w+)/$',
views.show_by_topic),
(r'^web/show_for_quiz/(?P<quiz>\d+)/$',
views.show_by_quiz),

(r'^web/show_for_author/(?P<author_username>\w+)/$',
views.show_by_author),

(r'^web/show_for_semester/(?P<semester>\w+)/$',
views.show_by_semester),

(r'^web/show_for_type/(?P<type>\w+)/$',
views.show_by_type),

#student portal TEST
(r'^portals/(?P<athena_id>\w+)/$',
student_views.student_portal),

# Student Landing Page
(r'^$', student_views.student_portal),

(r'^public/$', student_views.student_portal),

# Show a different video on the preview pane
(r'^(?P<topic_snippet_id>\d+)/$',
student_views.student_portal),

#Show a new topic assignment and specify
# the favorite toggled as "All" or "Favorites"
(r'^(?P<topic_snippet_id>\d+)/(?P<show>\w+)/$',
student_views.student_portal),

(r'^(?P<topic_snippet_id>\d+)/(?P<show>\w+)/
(?P<query_string>\w+)/$',
student_views.student_portal),

# tutorial problems
(r'^tutprobs/$', views.tutorial_main),
(r'^tutprobs/(?P<topic>.*)$', views.tutorial_by_topic),

(r'^tutprobs/(?P<topic>\w+)/
(?P<linked_problem_id>\d+)/$', views.tutorial_by_id),

)
```

A.3.2 Settings.py

```
# Django settings for usersite project.

DEBUG = True
TEMPLATE_DEBUG = DEBUG

ADMINS = (
    # ('Your Name', 'your_email@domain.com'),
)

AUTH_PROFILE_MODULE = 'tutorials.UserProfile'
## This is to associate each -User- with a
## -UserProfile- that stores the athena_id
## and favorites of the student
MANAGERS = ADMINS

DATABASE_ENGINE = 'sqlite3'
DATABASE_NAME = '/home/caitlinj/website/djcode/usersite.db'
# Or path to database file if using sqlite3.
DATABASE_USER = ''             # Not used with sqlite3.
DATABASE_PASSWORD = ''         # Not used with sqlite3.
DATABASE_HOST = ''             # Set to empty string for localhost. Not used
with sqlite3.
DATABASE_PORT = ''             # Set to empty string for default. Not used
with sqlite3.

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
TIME_ZONE = 'America/Chicago'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'en-us'

SITE_ID = 1

# If you set this to False, Django will make some optimizations so as not
# to load the internationalization machinery.
USE_I18N = True

# Absolute path to the directory that holds media.
# Example: "/home/media/media.lawrence.com/"
```

```
MEDIA_ROOT = '/home/caitlinj/website/djcode/usersite/media/'

# URL that handles the media served from MEDIA_ROOT. Make sure to use a
# trailing slash if there is a path component (optional in other cases).
# Examples: "http://media.lawrence.com", "http://example.com/media/"
MEDIA_URL = 'http://lecture.csail.mit.edu/site_media/'

# URL prefix for admin media -- CSS, JavaScript and images. Make sure to use a
# trailing slash.
#ADMIN_MEDIA_PREFIX = '/media/'
ADMIN_MEDIA_PREFIX = '/media/admin-media/'

# Make this unique, and don't share it with anybody.
SECRET_KEY = [OMITTED]

# List of callables that know how to import templates from various sources.
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.load_template_source',
    'django.template.loaders.app_directories.load_template_source',
    'django.template.loaders.eggs.load_template_source',
)

TEMPLATE_CONTEXT_PROCESSORS = (
    'django.core.context_processors.auth',
)

MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
)

ROOT_URLCONF = 'usersite.urls'

TEMPLATE_DIRS = (
    '/home/caitlinj/website/djcode/usersite/templates',
    '/home/caitlinj/website/djcode/usersite/templates/records',
    '/home/caitlinj/website/djcode/usersite/templates/tutorials',
    '/home/caitlinj/website/djcode/usersite/templates/6004_tutorial_probs',
    '/home/caitlinj/website/djcode/usersite/tutprobs',
    )

FIXTURE_DIRS = (
    '/home/caitlinj/website/djcode/usersite/tutorials/fixtures',
    )

INSTALLED_APPS = (
    'django_extensions',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'usersite.tutorials',
    'django.contrib.admin',
    'django_filters',
    'south',
)
```

A.3.3 **Admin.py**

```
from django.contrib import admin
from usersite.tutorials.models import *

class PublicVideoAdmin(admin.ModelAdmin):
    list_display=('type', 'title', 'file_name','author', 'semester',
'num_favorites', 'id', 'file')
    list_filter=('type', 'title', 'author', 'semester', 'id', 'file')
    fields=('type', 'semester', 'author',  'file',  'title')
    def num_favorites(self, obj):
        return obj.userprofile_set.count()
    pass

class ViewIntervalAdmin(admin.ModelAdmin):
    list_display=('user', 'ta', 'time', 'range')
    list_filter=('user', 'ta', 'time',)

class FavoriteAdmin(admin.ModelAdmin):
    list_display=('profile', 'ta', 'time')
    list_filter=('profile', 'ta', 'time')

class TopicAssignmentAdmin(admin.ModelAdmin):
    list_display=('topic', 'title', 'id', 'quiz', 'video')
    list_filter=('topic', 'title', 'id', 'quiz', 'video')

class LinkedWebPageAdmin(admin.ModelAdmin):
    list_display=('name', 'url', 'pointer_on_page', 'topic_assignment')
    list_filter=('name', 'url', 'topic_assignment')

admin.site.register(UserProfile)
admin.site.register(PublicVideo, PublicVideoAdmin)
admin.site.register(TopicAssignment, TopicAssignmentAdmin)
admin.site.register(Comment)
admin.site.register(ViewInterval, ViewIntervalAdmin)
admin.site.register(LinkedWebPage, LinkedWebPageAdmin)
admin.site.register(Favorite, FavoriteAdmin)
```

A.3.4 **Manage.py**

```
from django.core.management import execute_manager
try:
    import settings # Assumed to be in the same directory.
except ImportError:
    import sys
    sys.stderr.write("Error: Can't find the file 'settings.py' in the
directory containing %r. It appears you've customized things.\nYou'll have to
run django-admin.py, passing it your settings module.\n(If the file
settings.py does indeed exist, it's causing an ImportError somehow.)\n" %
__file__)
    sys.exit(1)

if __name__ == "__main__":
    execute_manager(settings)
```

A.4 Miscellaneous

A.4.1 Enums.py

```
QUIZ_CHOICES = (
    ('', '------'),
    (1, 'Quiz 1'),
    (2, 'Quiz 2'),
    (3, 'Quiz 3'),
    (4, 'Quiz 4'),
    (5, 'Quiz 5'),
    )

SEMESTER_CHOICES = (
    ('', '--All Terms--'),
    ('S11', 'Spring 2011'),
    ('F10', 'Fall 2010'),
    ('S10', 'Spring 2010'),
    ('F09', 'Fall 2009'),
    ('S09', 'Spring 2009'),
    ('F08', 'Fall 2008'),
    ('S08', 'Spring 2008'),
    ('F07', 'Fall 2007'),
    ('S07', 'Spring 2007'),
    ('F06', 'Fall 2006'),
    ('S06', 'Spring 2006'),
    ('F05', 'Fall 2005') )

VIDEO_CHOICES = (
    ('', '-- All Video Types --'),
    ('Lecture', 'Lectures'),
    ('Section', 'Recitations'),
    ('OldQuiz', 'Past Quiz Problems'),
    ('LabHint', 'Lab Hints'),
    ('TutProb', 'Tutorial Problems'),
    ('Concept', 'Conceptual Reviews') )

TOPIC_CHOICES = (
    ('', '-- All Topics --'),
    ('BasicsOfInformation', 'Basics of Information'),
    ('TheDigitalAbstraction', 'The Digital Abstraction'),
    ('CMOSTechnology', 'CMOS Technology'),
    ('GatesAndBooleanLogic', 'Gates And Boolean Logic'),
    ('SynthesisOfCombinationalLogic', 'Synthesis of Combinational Logic'),
    ('SequentialLogic', 'Sequential Logic'),
    ('FSMs', 'FSMs'),
    ('SynchronizationAndMetastability', 'Synchronization and Metastability'),
    ('Pipelining', 'Pipelining'),
    ('ModelsOfComputation','Models of Computation'),
    ('ProgrammableMachines', 'Programmable Machines'),
    ('MachineLanguage','Machine Language'),
    ('StacksAndProcedures','Stacks and Procedures'),
    ('BuildingTheBeta', 'Building the Beta'),
    ('MemoryHierarchy', 'Memory Hierarchy'),
    ('Caches','Caches'),
    ('VirtualMemory','Virtual Memory'),
    ('VirtualMachines','Virtual Machines and OS Issues'),
    ('DevicesInterruptsAndRealTime', 'Devices Interrupts and Real Time'),
    ('Semaphores','Semaphores'),
    ('PipelinedBeta','Pipelined Beta') )

TUTORIAL_PROBLEM_URLS = {
    'BasicsOfInformation': "info.html",
    'TheDigitalAbstraction': "digital.html" ,
    'CMOSTechnology': "cmos.html",
    'GatesAndBooleanLogic': "gate.html",
    'SynthesisOfCombinationalLogic': "logic.html",
    'SequentialLogic': "sequential.html",
    'FSMs': "fsm.html",
    'SynchronizationAndMetastability': "synchronization.html",
    'Pipelining': "pipeline.html",
    'ModelsOfComputation': "computation.html",
    'ProgrammableMachines': "progmach.html",
    'MachineLanguage': "machinelang.html",
    'StacksAndProcedures': "procedures.html",
    'BuildingTheBeta': "beta.html",
    'MemoryHierarchy': "memhierarchy.html",
    'Caches': "caches.html",
    'VirtualMemory': "vm.html",
    'VirtualMachines': "os.html",
    'DevicesInterruptsAndRealTime': "realtime.html",
    'Semaphores': "semaphores.html",
    'PipelinedBeta': "pipelinedbeta.html" }

TOPIC_NUMBERS = {
    'BasicsOfInformation': 0,
    'TheDigitalAbstraction': 1,
    'CMOSTechnology': 2,
    'GatesAndBooleanLogic': 3,
    'SynthesisOfCombinationalLogic': 4,
    'SequentialLogic': 5,
    'FSMs': 6,
    'SynchronizationAndMetastability': 7,
    'Pipelining': 8,
    'ModelsOfComputation': 9,
    'ProgrammableMachines': 10,
    'MachineLanguage': 11,
    'StacksAndProcedures': 12,
    'BuildingTheBeta': 13,
    'MemoryHierarchy': 14,
    'Caches': 15,
    'VirtualMemory': 16,
    'VirtualMachines': 17,
    'DevicesInterruptsAndRealTime': 18,
    'Semaphores': 19,
    'PipelinedBeta': 20}

TOPIC_LIST = [
    'BasicsOfInformation',
    'TheDigitalAbstraction',
    'CMOSTechnology',
    'GatesAndBooleanLogic',
    'SynthesisOfCombinationalLogic',
    'SequentialLogic',
    'FSMs',
    'SynchronizationAndMetastability',
    'Pipelining',
    'ModelsOfComputation',
    'ProgrammableMachines',
    'MachineLanguage',
    'StacksAndProcedures',
    'BuildingTheBeta',
    'MemoryHierarchy',
    'Caches',
    'VirtualMemory',
    'VirtualMachines',
    'DevicesInterruptsAndRealTime',
    'Semaphores',
    'PipelinedBeta']
```

A.4.2 Loader.py

```
from models import *
from django.db import models
from django.contrib.auth.models import User
from django.core.files import File
import os, fnmatch, re
from django.conf import settings
from models import PublicVideo


def make():


## ----- Students ----- ##

    def make_student(athena_id, first_name, last_name, student_id):
        student=User.objects.filter(username=athena_id)
        if not student.count():
            s=User.objects.create_user(athena_id, '%s@mit.edu'
                                       %(athena_id), student_id)
            s.first_name=first_name
            s.last_name=last_name
            s.is_staff=False
            p = s.get_profile()
            p.student_id=student_id
            s.save()
            p.save()
            return s

        else:
            s = student.get(username=athena_id)
            s.first_name=first_name
            s.last_name=last_name
            s.is_staff=False
            s.save()
            return s

    benbit=make_student('benbit', 'Ben', 'Bitdiddle', '900000001')
    aphacker=make_student('aphacker', 'Alyssa P.', 'Hacker', '900000002')
    chipahoy=make_student('chipahoy', 'Chip', 'Ahoy', '900000003')
    alogue=make_student('alogue', 'Anna', 'Logue', '900000004')
```

```
## ----- Staff Members ----- ##


    def make_staff(athena_id, first_name, last_name):
        staff=User.objects.filter(username=athena_id)
        if not staff.count():
            member=User.objects.create_user(athena_id, '%s@mit.edu'
                                               %(athena_id), first_name)
            member.first_name=first_name
            member.last_name=last_name
            member.is_staff=True
            member.save()
            return member
        else:
            member=staff.get()
            member.save()
            return member

    ward=make_staff('ward', 'Steve', 'Ward')
    cjt=make_staff('cjt', 'Chris', 'Terman')

    caitlinj=make_staff('caitlinj', 'Caitlin', 'Johnson')
    sneuman=make_staff('sneuman', 'Sabrina', 'Neuman')
    kelleyk=make_staff('kelleyk', 'Kevin', 'Kelley')

    sarina=make_staff('sarina', 'Sarina', 'Canelake')
    dcrowell=make_staff('dcrowell', 'David', 'Crowell')
    renminbi=make_staff('renminbi', 'Becky', 'Bianco')
    bbasham=make_staff('bbasham', 'Brian', 'Basham')
    colosimo=make_staff('colosimo', 'Joe', 'Colosimo')
    kasittig=make_staff('kasittig', 'Karen', 'Sittig')
    drews=make_staff('drews', 'Andrew', 'Shapiro')



## ----- Lecture ----- ##


    staff_list = User.objects.filter(is_staff=True)

    semesters=[tuple[0] for tuple in SEMESTER_CHOICES]
    vid_types=[tuple[0] for tuple in VIDEO_CHOICES]
    usernames=[user.username for user in staff_list]
```

```
print "semesters: %s" %(semesters)
print "usernames: %s" %(usernames)


def fs_location(auth, type, term):
    return os.path.join(settings.MEDIA_ROOT, auth, type, term)

def is_video(file_name):
    extension=os.path.splitext(file_name)[1]
    for type in VIDEO_TYPE_LIST:
        if extension==type:
            return True
    return False

def remove_dupes(directory, file):
    for extension in VIDEO_TYPE_LIST:
        stem = os.path.splitext(os.path.split(file)[1])[0]
        for other_file in os.listdir(directory):
            other_name=os.path.split(other_file)[1]
            [other_stem, other_extension]
             =os.path.splitext(other_name)[0:2]
            if re.match(stem, other_stem):
                if other_stem[-1]=='_' and other_extension==extension:
                    os.remove(os.path.join(directory, other_file))
                    return True

def already_there(author, type, semester, file):
    authors = PublicVideo.objects.filter(author=author)
    types = authors.filter(type=type)
    semesters = types.filter(semester=semester)
    return semesters.filter(file_name=file).count()

def load_video(author, type, semester, video_path):
    video_name=os.path.split(video_path)[1]
    if not already_there(author, type, semester, video_name):
    ## get the user object with this username to assign it
        user_obj = staff_list.get(username=author)
        vid = PublicVideo(author=user_obj, type=type, semester=semester,
                          file_name=video_name)
        fil = File(open(video_path, 'rb'))
        vid.file_name=video_name
        vid.file.save(vid.file_name, fil, save=False)
        vid.save()
        return vid
    ## look by staff/type/semester
    def find_media():
        for username in usernames:
            for vid_type in vid_types:
                for semester in semesters:
                    directory =fs_location(username, vid_type, semester)
                    ## ONLY CHECK EXISTING DIRECTORIES!! ##
                    if os.path.exists(directory):
                        ## look at each file in the directory to laod
                        ## after duplicates are removed so we don't
                        ## load in duplicate objects with the same file
                        for file in os.listdir(directory):
                        ## and see if it's a movie file
                            if is_video(file):
                                proposed_path = os.path.join(directory, file)
                                new_vid=load_video(username, vid_type,
                                semester, proposed_path)
                                new_vid.save()
                                remove_dupes(directory, file)

                                ## if we had to remove dupes, fix file name
                                new_vid.file_name=file
                                new_vid.save()
    print "Finding Media"
    find_media()


## ------ Topics ------ ##

    titles = {
        'L03.mov':'Lecture 3',
        'L04.mov':'Lecture 4',
        'L05.mov':'Lecture 5',
        'L06.mov':'Lecture 6',
        'L07.mov':'Lecture 7',
        'L08.mov':'Lecture 8',
        'L09.mov':'Lecture 9',
        'L10.mov':'Lecture 10',
        'L11.mov':'Lecture 11',
        'L14.mov':'Lecture 14',
        'L15.mov':'Lecture 15',
    'L16.mov':'Lecture 16',
    'L17.mov':'Lecture 17',
    'L18.mov':'Lecture 18',
    'L19.mov':'Lecture 19',
    'L20.mov':'Lecture 20',
    'L21.mov':'Lecture 21',
    'L22.mov':'Lecture 22',
    'L23.mov':'Lecture 23',
    'L24.mov':'Lecture 24',
    'S10_Q1_P3.mov': 'Static Discipline',
    'S10_Q1_P2.mov': 'Timothy Leary\'s Calendar App',
    'S10_S1_P4.mov': 'Deja Vu',
    'S10_Q2_P2-2.mov': 'Timothy Leary\'s Calendar App (continued)',
    }

def assign_titles():
    videos = PublicVideo.objects.all()
    for video in videos:
        if video.title == '':
            if video.file_name in titles.keys():
                video.title = titles[video.file_name]

assign_titles()


## temp topic matcher for testing and development
topic_assignments = {
    'L03.mov':['CMOSTechnology'],
    'L04.mov':['SynthesisOfCombinationalLogic'],
    'L05.mov':['SequentialLogic'],
    'L06.mov':['FSMs'],
    'L07.mov':['SynchronizationAndMetastability'],
    'L08.mov':['Pipelining'],
    'L09.mov':['Pipelining', 'ModelsOfComputation'],
    'L10.mov':['ProgrammableMachines'],
    'L11.mov':['MachineLanguage'],
    'L14.mov':['BuildingTheBeta'],
    'L15.mov':['MemoryHierarchy'],
    'L16.mov':['Caches'],
    'L17.mov':['VirtualMemory'],
    'L18.mov':['VirtualMachines'],
    'L19.mov':['DevicesInterruptsAndRealTime'],
    'L20.mov':['DevicesInterruptsAndRealTime'],
    'L21.mov':['Semaphores'],
    'L22.mov':['PipelinedBeta'],
    'L23.mov':['PipelinedBeta', 'Pipelining'],
    'L24.mov':['ProgrammableMachines'],
    'S10_Q1_P3.mov': ['TheDigitalAbstraction', 'CMOSTechnology'],
    'S10_Q1_P2.mov': ['BasicsOfInformation'],
    'S10_S1_P4.mov': ['TheDigitalAbstraction', 'GatesAndBooleanLogic'],
    'S10_Q2_P2-2.mov': ['BasicsOfInformation'],
    }

def match_topics():
    base_url = "http://6004.csail.mit.edu/currentsemester/tutprobs/"
    videos = PublicVideo.objects.all()
    for video in videos:
        name=os.path.split(video.file_name)[1]
        if name in topic_assignments.keys():
            topic_match=topic_assignments[name]
            for topic in topic_match:
                ta = TopicAssignment(video=video, topic=topic)
                ta.save()
                lp_string = u'%s' %(topic)
                #print "lp_string = %s\n" %(lp_string)
                lp_leaf=TUTORIAL_PROBLEM_URLS[lp_string]
                #print "lp_leaf = %s\n" %(lp_leaf)
                lp_url=u'%s%s' %(base_url, lp_leaf)
                lp = LinkedWebPage(name=topic, url=lp_url)
                lp.topic_assignment=ta
                lp.save()

print "Matching Topics"
match_topics()
```

A.4.3 **Filters.py**

```
import django_filters
from usersite.tutorials.models import *
from django import forms

class TopicAssignmentFilterSet(django_filters.FilterSet):
    class Meta:
        model=TopicAssignment
        list_filter=['topic']
        fields=['topic',
                'quiz',
                'video__author',
                'video__type',
                'video__semester',
                'num_student_favorites',
                'num_staff_favorites']
    def __init__(self, *args, **kwargs):
        super(TopicAssignmentFilterSet, self).__init__(*args, **kwargs)
```

A.4.4 Forms.py

```
from django.shortcuts import render_to_response
from django.forms import ModelForm
from django.http import HttpResponseRedirect
from django import forms
from usersite.tutorials.models import Comment, PublicVideo

class CommentForm(ModelForm):
    fields=['text']
    # want to auto populate video and username fields
    class Meta:
        model=Comment

class PublicVideoForm(ModelForm):
    file=forms.FileField()
    class Meta:
        model=PublicVideo
        # author should be auto-assigned but pre-populated
        fields=['author','title', 'type', 'semester', 'file', 'description']
```

Appendix B: HTML Templates

B.1 Base Templates

B.1.1 Base.html

```
{% block header %} {% endblock %}
{% include "masthead.html" %}
{% block content %} {% endblock %}
```

B.1.2 Two_column.html

```
{% extends "base.html" %} {% block content %}
{% block page_top %} {% endblock page_top %}
{% block main_column %} {% endblock main_column %}
{% block sidebar %} {% endblock sidebar %}
{% endblock content %}
```

B.1.3 Three_block.html

```
{% extends "base.html" %} {% block content %}
{% block main_column %} {% endblock main_column %}
{% block browse_sidebar %} {% endblock browse_sidebar %}
{% block bottom_block %} {% endblock bottom_block %}
{% endblock content %}
```

B.2 Student Landing Page

B.2.1 Browse.html

```
{% extends "three_block.html" %}

{% block title %}Course Media{% endblock %}

{% block header %}
      {% include "interval_movie_header.html" %}
{% endblock header %}

{% block browse_sidebar %}
      {% include "student_browse.html" %}
{% endblock browse_sidebar %}

{% block main_column %}
      {% include "movie_div.html" %}
{% endblock main_column %}

{% block bottom_block %}
      {% include "mbrowser.html" %}
{% endblock bottom_block %}
```

B.2.2 **Student_browse.html**

```
<div id="sidebar_content">

  {% ifequal show 'Favorites' %}
  Displaying your favorite videos.
  <br>
  <a href="/{{selected_ta.id}}/All/"> [Show All Clips]</a>

  {% else %}
  Displaying all videos.
  <br>
  <a href="/{{selected_ta.id}}/Favorites/"> [Show Your Favorites] </a>
  {% endifequal %}
  {% if user.is_staff %}
  <p>
  <a href="/upload_video/">Upload New Video</a> and Assign Topic
  <br />
  <p>
  <a href="/select_video_for_assignment/">Select Existing Video</a> for Topic
Assignment
  {% endif %}

</div>
```

B.2.3 Mbrowser.html

```
<table border="1" class="mbrowser"
      style="display:block;clear:both">
  <tr>
        <th>Title</th>
        <th>Topic</th>
        <th>Quiz #</th>
        <th>Type</th>
        <th>Author</th>
        <th>Semester</th>
        <th>Student Favorites</th>
        <th>Staff Favorites</th>
        <th>Preview</th>
        <th>Detail View</th>
  </tr>
  <tr>
      <td> -- </td>
     <form action="" method="get">
      <td>{{topic_assignment_filterset.form.topic}}</td>
      <td>{{topic_assignment_filterset.form.quiz}}</td>
      <td>{{topic_assignment_filterset.form.video__type}}</td>
      <td>{{topic_assignment_filterset.form.video__author}}</td>
      <td>{{topic_assignment_filterset.form.video__semester}}</td>
      <td></td>
      <td></td>
      <td colspan="2" style="text-align:center">
        <input style="font-weight:bold" type="submit"
               value="-- Apply Filters--"/></td>
     </form>
  </tr>

  {% for ta in topic_assignment_filterset reversed %}
  <tr>
        <td>
        {% if ta.title %}
        <a href="/web/show_media/{{ta.id}}/">{{ta.title}}</a>
        {% else %}
        <a href="/web/show_media/{{ta.id}}/">untitled # {{ta.id}}
        </a>
        {% endif %}
        </td>
        <td>
            <a href="/web/show_for_topic/{{ta.topic}}/">
                    {{ta.topic}}</a></td>
        <td>
            <a href="/web/show_for_quiz/{{ta.quiz}}/">
                    {{ta.quiz_verbose}}</a></td>
        <td>
            <a href="/web/show_for_type/{{ta.video.type}}/">
                    {{ta.video.type}}</a></td>
        <td>
            <a href="/web/show_for_author/
            {{ta.video.author.username}}/">
                    {{ta.video.author}}</a></td>
        <td>
            <a href="/web/show_for_semester/
            {{ta.video.semester}}/">
                    {{ta.video.semester}}</a></td>
        <td>{{ta.num_student_favorites}}</td>
        <td>{{ta.num_staff_favorites}}</td>
        <td style="text-align:center">
            <a href="/{{ta.id}}/{{query_string}}">
            <img style="margin:0;padding:0;height:20px"
             src="/site_media/images/eye.png"
             height="10"></img></a>
        </td>
        <td style="text-align:center">
             <a href="/web/show_media/{{ta.id}}/">
             <img style="margin:0;padding:0;height:20px;"
             src="/site_media/images/arrow.jpg"
             height="10"></img></a>
        </td>
  </tr>
  {% endfor %}
</table>
```

B.3 Single-video Player

B.3.1 Show_media.html

```
{% extends "two_column.html" %}

{% block title %}Course Media{% endblock %}

{% block header %}
{% include "interval_movie_header.html" %}
{% endblock header %}

{% block main_column %}
{% include "comment_view.html" %}
{% include "timing_fields.html" %}
{% endblock main_column%}

{% block sidebar %}
{% include "similar_video_bar.html" %}
{% include "favorite_button.html" %}
{% endblock sidebar %}
```

B.3.2 Similar_video_bar.html

```
<p class="sidebar_header">View more Videos with the same:</p>

<ul>
  <li>Topic:
    <a href="/web/show_for_topic/{{selected_ta.topic}}/">
      {{selected_ta.topic}}</a></li>

  <li>Author:
    <a href="/web/show_for_author/{{selected_ta.video.author.username}}/">
      {{selected_ta.author.get_full_name}}</a></li>

  <li>Semester:
    <a href="/web/show_for_semester/{{selected_ta.video.semester}}/">
      {{selected_ta.video.semester}}</a></li>

  <li>Type:
    <a href="/web/show_for_type/{{selected_ta.video.type}}/">
      {{selected_ta.video.type}}</a>
  </li>
</ul>


<p class="sidebar_header">Tutorial Problems for this Topic:</p>

<ul>
  {% for linked_problem in linked_problems %}

  <li> <a href="/tutprobs/{{topic}}/{{linked_problem.id}}/">
      {{linked_problem.name}}</a></li>

  {% endfor %}

</ul>
```

B.3.3 Favorite_button.html

```
<div name="favorite_div" id="favorite_div">
  <table>
    <tr>
      <td class="sidebar_header" style="display:inline">
              <form action="" method="post" name="change_favorite"
                       id="change_favorite">
                <input type="hidden" name="username" id="username"
                       value="{{user.username}}" />
                <input type="hidden" name="ta_id" id="ta_id"
                       value="{{selected_ta.id}}" />
                {% if user.is_authenticated and is_user_favorite %}
                <!-- This video is marked as one of your favorites.-->
                <input type="submit" id="submit_favorite"
                       name="submit_favorite" value="Remove Favorite" />
                {% endif %}

                {% if user.is_authenticated and not is_user_favorite %}
                <!-- You have not yet added this video to your favorites.-->
                <input type="submit" id="submit_favorite"
                       name="submit_favorite" value="Add Favorite"
                       style="align:right"/>
                {% endif %}
              </form>
      </td></tr>
  </table>
</div>

<script type="text/javascript"
src="/site_media/comment_submission.js"></script>
<script type="text/javascript" src="/site_media/jquery.js"></script>
<script type="text/javascript">
$('#change_favorite').submit(
function() {
change_favorite();
console.log("here");
return false; }
);
</script>
```

B.3.4 Timing_fields.html

```
<table>
  <tr>
    <td>
      <input type="hidden" id="interval_start_units"
             name="interval_start_units"/>
      <input type="hidden" id="interval_start_display"
             name="interval_start_display" value="0"/>
      <input type="hidden" id="interval_pause_units"
             name="interval_pause_units"/>
      <input type="hidden" id="interval_pause_display"
             name="interval_pause_display" value="0"/>
      <input type="hidden" id="interval_skip_units"
             name="interval_skip_units"/>
      <input type="hidden" id="interval_skip_display"
             name="interval_skip_display" value="0"/>
      <input type="hidden" id="play_timer" name="play_timer" value="?"/>
      <input type="hidden" id="play_timer_id" name="play_timer_id" value="?"/>
      <input type="hidden" id="start_seconds_display"
             name="start_seconds_display" value="?"/>
      <input type="hidden" id="pause_seconds_display"
             name="pause_seconds_display" value="?"/>
      <input type="hidden" id="skip_seconds_display"
             name="skip_seconds_display" value="?"/>
      <input type="hidden" id="start_timer" name="start_timer" value="0"/>
      <input type="hidden" id="start_timer_id" name="start_timer_id"/>
    </td>
  </tr>
</table>

<table>
  <input type="hidden" id="timescale" name="timescale" value="?"/>
  <form id="iform" method="post">
    <input type="hidden" id="ta_id" name="ta_id" value="{{selected_ta.id}}"/>
    <input type="hidden" id="user" name="user" value="{{user}}"/>
    <input type="hidden" id="iform_start_time" name="iform_start_time"
           value="0"/>
    <input type="hidden" id="iform_end_time" name="iform_end_time" value="0"/>
  </form>
</table>
```

B.3.5 Comment_view.html

```
<table border="0"  id="comment_box">
       {% if user.is_authenticated %}
       <form id="comment_form" name="comment_form" action="" method="post">
             <tr>
                 <td colspan="2">
                      <textarea id="comment_textarea" rows="5" name="text">
                      Tell us what you thought of the video! Was it too long?
                      Was one part particularly helpful?
                      Leave your comments here...
                      </textarea>
                 </td>
             </tr>
             <tr>
                 <td>
                 Visibility:
                 <select name="permissions">
                 <option value="students"> All Students</option>
                 <option value="staff">
                 Staff Members and {{user.username}}
                 </option>
                 </select>
                   </td>
                   <td style="text-align:right">
                        <input type="hidden" name="username" id="username"
                        value="{{user.username}}" />
                        <input type="hidden" name="ta_id" id="ta_id"
                        value="{{selected_ta.id}}" />
                        <input type="submit" name="submit" id="submit"
                        value="Submit Comment" />
                   </td>
                 </tr>
             </form>
             {% endif %}
</table>

<table id="comment_display" class="comment_display">
             {% if comments.count %}
             <tr>
               <td colspan="2" class="sidebar_header">
                 Comments about this video:
               </td>
             </tr>
```

```
              {% endif %}

              {% for comment in comments %}
              <tr>
                <td colsapn="2">
                  "{{comment.text}}"
                  <br/> -- {{comment.user.username}} at {{comment.time}}
                </td>
              </tr>
              {% endfor %}
</table>
<script type="text/javascript"
src="/site_media/comment_submission.js"></script>
<script type="text/javascript" src="/site_media/jquery.js"></script>
<script type="text/javascript">
<!-- make sure Jquery is loaded somewhere else,
like in interval_movie header.html or movie_header.html -->
$('#comment_form').submit(
  function() {
    submit_comment();
    return false;
  }
);

</script>
```

B.3.6 Display_interval_views.html

```
{% extends "two_column.html" %}

{% block title %}Course Media{% endblock %}

{% block header %}

{% include "interval_movie_header.html" %}

{% endblock header %}

{% block body_attributes %}onload="RegisterListeners();"{% endblock
body_attributes %}


{% block main_column %}


<table border="0">


  <tr>
    <td>

<!--
      <div id="movie_div" style="float:left;align:left"></div>

      <script type="text/javascript">
              document.getElementById('movie_div').innerHTML = qtEmbed;
      </script>
-->
{% include "movie_div.html" %}

    </td>
  </tr>

  <!-- 800 by 300 graph in same table -->
    <tr>
    <td>
      <div id="view_graph_div" class="view_graph_div"
           style="float:left;align:left">
      <img id="view_graph" name="view_graph" src="{{img_url}}" />
      </div>
    </td>
  </tr>

  <tr>
    <td>
      {% include "comment_view.html" %}
    </td>
  </tr>

</table>

{% endblock main_column%}


{% block sidebar %}
<!--
{% include "similar_video_bar.html" %}
-->
{% include "favorite_button.html" %}


{% endblock sidebar %}
```

B.4 Authentication and Password Changes

B.4.1 Login.html

```
{% extends "base.html" %}

{% block content %}

{% if form.errors %}
<p class="error"> Sorry, that's not a valid username or password </p>
{% endif %}

<form action="" method="post">
  <label for="username">User name:</label>
  <input type="text" name="username" value="" id="username">

  <label for="password">Password:</label>
  <input type="password" name="password" value="" id="password">

  <input type="submit" value="login" />
  <input type="hidden" name="next" value="/accounts/profile/" />
</form>
{% endblock %}
```

B.4.2 Password_change_form.html

```
{% extends "base.html" %}

{% block content %}

<!-- Content -->
<div id="content" class="colM">

  <h1>Password change</h1>

  <p>Please enter your old password, for security's sake,
  and then enter your new password twice so we can verify
  you typed it in correctly.</p>

  <form action="" method="post">
    <p class="aligned wide">
              <label for="id_old_password">Old password:</label>
              <input type="password" name="old_password"
                 id="id_old_password" /></p>
    <p class="aligned wide">
              <label for="id_new_password1">New password:</label>
              <input type="password" name="new_password1"
                 id="id_new_password1" /></p>
    <p class="aligned wide">
              <label for="id_new_password2">Confirm password:</label>
              <input type="password" name="new_password2"
                 id="id_new_password2" /></p>
    <p><input type="submit" value="Change my password" /></p>
  </form>
  <br class="clear" />
</div>
    <!-- END Content -->

{% endblock %}
```

B.4.3 **Password_change_done.html**

```
{% extends "base.html" %}

{% block content %}

<!-- Content -->
<div>
<p>
  <h2>
    Change Successful! Please click one of the navigational links above to
continue.
  </h2>
  <br>
</div>
    <!-- END Content -->

{% endblock %}
```

B.5 Video Upload and Chapter Assignment

B.5.1 Upload_video.html

```
{% extends "base.html" %}

{% block title %} Upload a Public Video{% endblock %}

{% block content %}
<form action="" enctype="multipart/form-data" method="POST">
  <fieldset>
    <legend>
      Welcome {{user.username}}.
                Please Select a File to upload and fill out the
                relevant fields.
                <br />
                You can, alternatively, select an existing video
                for topic assignment from the menu at the bottom
                of the page.
                <br />
                You will then be directed to a page where you
                can set topic segments.
    </legend>

    {{video_form.as_p}}
  </fieldset>
  <input type="submit" value="Save and Proceed to Topic Assignment"/>
</form>

<br />
<br />

{% include "video_select.html" %}

{% endblock %}
```

B.5.2 Topic_assignment.html

```
{% extends "two_column.html" %}

{% block title %}
{{title_string}}
{% endblock %}

{% block page_top %}
{{header_string}}
{% endblock page_top %}

{% block main_column %}
<table border="1">
  <tr>
    <th>
      Video
    </th>
    <th>
      # of Staff Faves
    </th>
    <th>
      # of Student Faves
    </th>
  </tr>

  {% for ta in ta_query_set %}
  <tr>
    <td>
      <a href="/web/show_media/{{ta.id}}/">
      {% if ta.video.title %}
      {{ta.video.title}}
      {% else %}
      {{ta.video.file_name}}
      {% endif %}
      </a>
    </td>
    <td>
      {{ta.get_num_staff_favorites}}
    </td>
    <td>
      {{ta.get_num_student_favorites}}
    </td>
  </tr>
  {% endfor %}
</table>
{% endblock main_column %}
{% block sidebar %}

<p class="sidebar_header" align="center">
  <a href="/accounts/profile/">
    Return to Media Browser</a></p>

{% endblock sidebar %}
```

Appendix C: JavaScript

C.1 QuickTime Files

C.1.1 Common_quicktime_methods.js

```
// methods that depend on document.movie1
function playhead_position() {
    return document.movie1.GetTime();
}

function time_scale() {
              return document.movie1.GetTimeScale();
}

// internal methods
function set_display_area_to_fit_movie()
{
    var obj = document.movie1;
    var rectangle = obj.GetRectangle();

              if (rectangle)
              {
              rectangle = rectangle.split(',');
              var x1 = parseInt(rectangle[0]);
              var x2 = parseInt(rectangle[2]);
              var y1 = parseInt(rectangle[1]);
              var y2 = parseInt(rectangle[3]);

              var width = (x1 < 0) ? (x1 * -1) + x2 : x2 - x1;
              var height = (y1 < 0) ? (y1 * -1) + y2 : y2 - y1;
              }
              else
              {
              // a mov containing only audio
              var width = 200;
              var height = 0;
              }

  height_field = document.getElementById('video_height');
  width_field  = document.getElementById('video_width');

  if (height_field)
                document.getElementById('video_height').value = height;

  if (width_field)
                document.getElementById('video_width').value = width;

              obj.width = width;
              obj.height = height + 16;

              obj.SetControllerVisible(true);
}

function format_time(time_in_video_units){
              totalSec = time_in_video_units / time_scale();

              hours = parseInt( totalSec / 3600 ) % 24;
              minutes = parseInt( totalSec / 60 ) % 60;
              seconds = parseInt( totalSec ) % 60;
              fframes  = Math.round( ((totalSec % 60) - seconds) * 100);

              result =  zero_pad(hours) + ":" + zero_pad(minutes) + ":" +
zero_pad(seconds) + ":" + zero_pad(fframes);
              return result;
};

function zero_pad(number)
{
              return (number < 10 ? "0" + number: number)
}

function myAddListener(obj, evt, handler, captures)
{
              if ( document.addEventListener )
              obj.addEventListener(evt, handler, captures);
              else
              // IE
              obj.attachEvent('on' + evt, handler);
}


function RegisterListener(eventName, objID, embedID, listenerFcn)
{
    //console.log("register_listener { event : " + eventName + " listenerFcn :
" + listenerFcn);
            var obj = document.getElementById(objID);
            if ( !obj )
            obj = document.getElementById(embedID);
            if ( obj )
            myAddListener(obj, eventName, listenerFcn, false);
}
```

C.1.2 Interval_movie_header.html

```
<script type="text/javascript" src="/site_media/AC_QuickTime.js"></script>
<script type="text/javascript"
src="/site_media/common_quicktime_methods.js"></script>
<script type="text/javascript" src="/site_media/jquery.js"></script>
<script type="text/javascript"
src="/site_media/staff_interval_methods.js"></script>

<script type="text/javascript">

  // define the video here
  var qtEmbed = QT_GenerateOBJECTText_XHTML(
              '{{selected_ta.video.get_absolute_url}}',
              '600', // width
              '475', // height: set this to actual height
                 // + 20 (to leave space for controller)
              '',   // required blank field
              'enablejavascript', 'true',
              'obj#id', 'movie1',
              'emb#name', 'movie1',
              'emb#id', 'movie1_emb',
              'postdomevents', 'true',
              'autoplay', 'false',
              'controller', 'true',
              'scale', 'aspect'
  )

  // called from body.onload to set up listeners that will wait for quicktime
  // movie to send events

  function RegisterListeners()
  {
  // when the movie loads,
  // it will set the end time to the duration of the movie
  RegisterListener('qt_loadedmetadata', 'movie1',
                     'movie1_emb', setup_movie);
  RegisterListener( 'qt_loadedmetadata', 'movie1',
                     'movie1_emb', set_times_and_play_movie);
  RegisterListener( 'qt_canplay', 'movie1',
                     'movie1_emb', setup_seconds);
  RegisterListener( 'qt_play', 'movie1',
                        'movie1_emb', check_and_send);
    RegisterListener( 'qt_timechanged', 'movie1',
                        'movie1_emb', pause_and_start);
    RegisterListener( 'qt_stop', 'movie1',
                        'movie1_emb', set_interval_pause);
    RegisterListener( 'qt_pause', 'movie1',
                        'movie1_emb', set_interval_pause);

    }



function set_times_and_play_movie()
{
    var timescale = document.movie1.GetTimeScale();
    console.log("(set_times_and_play_movie) timescale
              = " + timescale);
    document.getElementById('timescale').value = timescale;
    var start_time = {{selected_ta.start_time}};
    var end_time   = {{selected_ta.stop_time}};
    console.log("(set_times_and_play_movie) start_time
              = " + start_time);
    console.log("(set_times_and_play_movie) end_time
              = " + end_time);

    if(typeof(start_time) == "undefined")
              {
                  start_time = 0;
              }
    if(typeof(end_time) == "undefined")
              {
                  end_time = document.movie1.GetDuration();
              }

    if(end_time == 0)
              {
                  var length =  document.movie1.GetDuration();
                  console.log("setting end_time to " + length);
                  end_time = length;
              }

    set_start_and_end_times(start_time, end_time);
    //document.movie1.Play();
}
```

```
function set_start_and_end_times(start_time, end_time)
{

    console.log("(set_start_and_end_times)");
    document.movie1.SetStartTime(start_time);
    document.movie1.SetEndTime(end_time);
}


function setup_movie()
{
  console.log("(setup_movie)");
  // to check that the movie meta data has been loaded sufficiently
  var movie_length = document.movie1.GetDuration().value;
  console.log("(setup_movie) movie duration = " + movie_length);
//  set_display_area_to_fit_movie();

}



</script>
```

C.2 jQuery/Ajax

C.2.1 Comment_submission.js

```
function change_favorite()
{
    console.log("in change_favorite");
    var username = $('input[name=username]').val();
    console.log("username");
    var ta_id = $('input[name=ta_id]').val();
    console.log("ta_id");
    var button_value = $('input[name=submit_favorite]').val();
    var is_favorite = true;
    $.ajax({    type: 'POST',
              url: "/add_favorite/",
              data: { username: username, ta_id: ta_id,
              button_value: button_value },
              dataType: "json",
              success: function(response)
              {
                  console.log("in change favorite success function");
                  document.getElementById('submit_favorite').value
              = response.new_button_text;
              },
              });

}

function submit_comment()
{
    var username = $('input[name=username]').val();
    var ta_id = $('input[name=ta_id]').val();
    var text = $('textarea[name=text]').val();
    var permissions = $('select[name=permissions]').val();
    console.log("in submit_comment");
    console.log("username = " + username +
              ", permissions = " + permissions +
              ", ta_id = " + ta_id + ", text = " + text);
    $.ajax({    type: 'POST',
              url: "/comment_update/",
              data: { username : username,
                    text : text,
                    permissions : permissions,
                    ta_id : ta_id, },
            success: function(response){
            console.log("in success function for submit_comment");
                var new_comment = "<tr><td colsapn=\"2\">" +
              response.text + "<br/> --" +
              response.username + " at " +
              response.time +"</td></tr>";
                $('.comment_display').append(new_comment);
                    },
            dataType: "json",
            });
    return false;
    // this is supposed to prevent the default submission behavior
}
```

C.2.2 Favorite_button.js

```
<div name="favorite_div" id="favorite_div">
  <table>
    <tr>
      <td class="sidebar_header" style="display:inline">
              <form action="" method="post" name="change_favorite"
                    id="change_favorite">
                <input type="hidden" name="username" id="username"
                       value="{{user.username}}" />
                <input type="hidden" name="ta_id" id="ta_id"
                       value="{{selected_ta.id}}" />

                {% if user.is_authenticated and is_user_favorite %}
                <!-- This video is marked as one of your favorites.-->
                <input type="submit" id="submit_favorite"
                       name="submit_favorite" value="Remove Favorite" />
                {% endif %}

                {% if user.is_authenticated and not is_user_favorite %}
                <!-- You have not yet added this video to your favorites.-->
                <input type="submit" id="submit_favorite"
                       name="submit_favorite" value="Add Favorite"
                       style="align:right"/>
                {% endif %}
              </form></td></tr>
  </table>
</div>

<script type="text/javascript"
src="/site_media/comment_submission.js"></script>
<script type="text/javascript" src="/site_media/jquery.js"></script>
<script type="text/javascript">
$('#change_favorite').submit(
function() {
change_favorite();
console.log("here");
return false;
  }
);
</script>
```

C.2.3 Interval_methods.js

```
function setup_seconds()
{
    var tscale = time_scale();
    document.getElementById('timescale').value = tscale;

    document.getElementById('start_seconds_display').value = 0;
    document.getElementById('pause_seconds_display').value = 0;
    document.getElementById('start_timer').value = 0;
}


function clear_start_timer()
{
    var timer_id = document.getElementById('start_timer_id').value;
    console.log("clearing start timer");
    document.getElementById('start_timer').value = 0;
    clearInterval(timer_id);
}


function stop_start_timer()
{
    var current_value = document.getElementById('start_timer').value;
    console.log("stopping the start timer. current value = " + current_value);
    // doesn't clear the displayed value or the timer value
    // only stops the timer. Need to wait until after
    // check_and_send to clear the timer (reset to 0)
    var timer_id = document.getElementById('start_timer_id').value;
    clearInterval(timer_id);
}


function get_current_second()
{
    var units = playhead_position();
    var timescale = parseInt(document.getelementById('timescale')
                .value);
    return Math.floor(units/timescale);
}


//interval start
```

```
function increment_start_timer()
{
    var timer = document.getElementById('start_timer').value;
    var timer_value = parseInt(timer);
    var timer_id = document.getElementById('start_timer_id').value;
    console.log("in increment_start_timer with timer #" + timer_id
              + " " + timer_value);
    timer_value = timer_value + 1;
    document.getElementById('start_timer').value = timer_value;
}

function begin_start_timer(init)
{
    var timer_id = setInterval("increment_start_timer()", 1000);
    console.log("(begin_start_timer) create the start_timer with id ="
              + timer_id);
    document.getElementById('start_timer_id').value = timer_id;
    console.log("(begin_start_timer) value of init = " + init);

    //var timer_id = document.getElementById('start_timer_id').value;
    document.getElementById('start_timer').value = init;
    var last_start_time =
              document.getElementById('start_timer').value;
    console.log("(begin_start_timer) start_timer set to = "
              + last_start_time);
    //increment_start_timer();
}

function set_interval_start()
{
    var start_units = playhead_position();
    document.getElementById('interval_start_units')
                .value = start_units;
    var start = format_time(start_units);
    document.getElementById('interval_start_display').value = start;
    var timescale = document.getElementById('timescale').value;
    var start_seconds = Math.floor(start_units/timescale);
    console.log("(set_interval_start) start_seconds = "
              + start_seconds);
    document.getElementById('start_seconds_display')
                .value = start_seconds;
}

// form to send interval to server
```

```
function check_and_send()
{
    var last_start_time =
              parseInt(document.getElementById('start_seconds_display').value)
;
    var interval_length =
              parseInt(document.getElementById('start_timer').value);
    var last_pause =
              document.getElementById('pause_seconds_display').value;

    /*
       Note: it's necessary to use parseInt because of string addition
       that will happen if we dont' convert them to ints first
    */

    var current_position = playhead_position();
    var timescale = document.getElementById('timescale').value;
    var current_second = Math.floor(current_position/timescale);

    console.log("(check_and_send) last_start_time = "
              + last_start_time);
    console.log("(check_and_send) interval_length = "
              + interval_length);
    var last_watched_second = last_start_time + interval_length;
    console.log("(check_and_send) laast_watched_second = "
              + last_watched_second);
    console.log("(check_and_send) with current_second = "
              + current_second);

    // if the latter is true, we need to package
    // up a new interval and send a message to server
    if ( (last_start_time + interval_length) < current_second)
              {
                  console.log("passed the if statement....");
                  console.log("--last_start_time = " + last_start_time);
                  console.log("--interval_length = " + interval_length);
                  $.ajax({
                  type:'POST',
              url: "/post_test/",
              data: { iform_start : last_start_time,
                  iform_end : last_pause,
                  user : document.getElementById('user').value,
                  ta_id : document.getElementById('ta_id').value,
                  },
              success: function(responseData)
              {
                  console.log(responseData);
                  //alert(responseData);
              },
              dataType: "text"
              });
                  set_interval_start();
                  begin_start_timer(0);
              }
    else {
              var last_start = document.getElementById('start_timer').value;
              console.log("in check and send, starting the timer again at "
              + last_start);
              begin_start_timer(last_start);
    }
    //set_interval_start();
    // start the timers again
}


// interval pause
function set_interval_pause()
{
    var pause_units = playhead_position();
    document.getElementById('interval_pause_units')
                 .value = pause_units;
    var pause = format_time(pause_units);
    document.getElementById('interval_pause_display').value = pause;
    var timescale = document.getElementById('timescale').value;
    var pause_seconds = Math.floor(pause_units/timescale);
    console.log("(set_interval_pause) pause_seconds = "
              + pause_seconds);

    document.getElementById('pause_seconds_display')
                 .value = pause_seconds;
    stop_start_timer();
}
```

Appendix D: CSS Profiles

D.1 Main Style Sheet Definitions

D.1.1 Usersite.css

```
body
{
    margin:0;
    padding:0;
    background-color:#5E778E;
}

td, th
{
   line-height:105%;
   max-width:220px;
}

.smtd
{
   max-width:100px;
}


div, p, ul, ol
{
    font-family:Arial, Helvetica, sans-serif;
    font-size:100%;

}


ul, ol
{
    margin-top:0.5em;
}

#base
```

```
{
    height:100%;
    background-color:#5E778E;
}

#top_block
{
    position:relative;
    padding:0.5em;
    width:90%;
    height:70%;
}

#main_column
{
    background-color:#99AABB;
    padding:1%;
    width:600px;
    position:absolute;
    left:0;
}

#sidebar
{
    font-size:75%;
    width:30%;
    float:right;
    margin-left:3em;
    position:relative;
    background-color:#AABBCC;
    padding:1%;
    display:table-column;
}

#browse_sidebar
{
    position:absolute;
    top:0;
    left:700px;
    float:right;
    padding:1em;
    background-color:#CCCCCC;
    margin:10px;
    width:150px;
}

#sidebar_content
{
    font-size:16px;
    display:block;
    height:100%;
}


#bottom_block
{
    background-color:#FAFAFF;
    margin-top:0.5em;
    display:inline;
    position:absolute;
    top:575px;
    clear:top;
}

.clear
{
    clear:both;
    height:0;
    font-size:0;
}

.sidebar_header
{
    font-weight:bold;
    font-size:120%;
    margin-bottom:0px;
}

#masthead
{
    margin:10px;
    height:15px;
    width:auto;
    font-size:65%;
    padding:.5em 2em;
    background-color:#CCCCCC;
    border-bottom:1px solid black;
}
```

```
#content_container
{
    width:90%;
    display:block;
    margin:auto;
    margin-top:1em;
}

#home_nav
{
    display:inline;
    float:left;
    text-align:left;
}

#welcome_message
{
    display:inline;
    float:right;
    text-align:right;
    margin-left:3em;
    margin-right:5em;
}

#page_top
{
    font-size: 125%;
    font-weight:bold;
    text-decoration:underline;
    margin-bottom:10px;
    width:80%

}


.movie_div
{
    width:80%;
    margin:10%;
}
```

```
.comment_box
{
    display:block;
    float:left;
    margin:2%;
    width:100%;
}

.comment_textarea
{
    width:90%;
}

.mbrowser
{
    background-color:#FAFAFF;
    font-size:80%;
    top:10px;
    float:left;
    margin:0;
    position:absolute;
}

.mast_link
{
    font-size:120%;
    font-weight:bold;
    padding:0.5em;
}
```

D.1.2 **Default.css**

```
/*
Design by Free CSS Templates
http://www.freecsstemplates.org
Released for free under a Creative Commons Attribution 2.5 License
*/

body {
		margin: 0;
		padding: 0;
		background: #28313A url(images/img01.jpg) repeat-x left top;
		font-size: 12px;
		font-family: Georgia, "Times New Roman", Times, serif;
		text-align: justify;
		color: #5C5C5C;
}

h1, h2, h3 {
		margin: 0;
		text-transform: lowercase;
		font-weight: normal;
		color: #FFFFFF;
}

h1 {
		letter-spacing: -1px;
		font-size: 32px;
}

h2 {
		font-size: 23px;
}

p, ul, ol {
		margin: 0 0 2em 0;
		text-align: justify;
		line-height: 26px;
}

a {
		color: #1B75A9;
}
```

```
a:hover, a:active {
               text-decoration: none;
               color: #1B75A9;
}

a:visited {
               color: #1B75A9;
}

img {
               border: none;
}

img.left {
               float: left;
               margin-right: 15px;
}

img.right {
               float: right;
               margin-left: 15px;
}

/* Form */

form {
               margin: 0;
               padding: 0;
}

fieldset {
               margin: 0;
               padding: 0;
               border: none;
}

legend {
               display: none;
}

input, textarea, select {
               font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
               font-size: 13px;
```

```
              color: #333333;
}

#wrapper {
              margin: 0;
              padding: 0;
}

/* Header */

#header {
              width: 880px;
              margin: 0 auto;
              height: 60px;
              border: 10px #FFFFFF solid;
}

/* Menu */

#menu {
              float: left;
              width: 880px;
              height: 58px;
              background: url(images/img02.jpg) repeat-x left top;
}

#menu ul {
              margin: 0;
              padding: 23px 0 0 20px;
              list-style: none;
              line-height: normal;
}

#menu li {
              float: left;
              text-align: center;
}

#menu a {
              display: block;
              padding: 0 50px;
              background: url(images/img03.jpg) no-repeat right 50%;
              text-decoration: none;
              text-transform: uppercase;
              font-size: 11px;
              color: #FFFFFF;
}

#menu a:hover {
              color: #FFFFFF;
}

#menu .current_page_item a {
              color: #FFFFFF;
}

/** LOGO */

#logo {
              width: 880px;
              height: 130px;
              margin: 0 auto;
}

#logo h1, #logo h2 {
              float: left;
              margin: 0;
              padding: 50px 0 0 0px;
              line-height: normal;
}

#logo h1 {
              font-family: Georgia, "Times New Roman", Times, serif;
              font-size:40px;
}

#logo h1 a {
              text-decoration: none;
              color: #28313A;
}

#logo h1 a:hover { text-decoration: underline; }

#logo h2 {
              float: left;
              padding: 65px 0 0 18px;
              font-family: Georgia, "Times New Roman", Times, serif;
              font-size: 25px;
              color: #28313A;
}

#logo p a {
              text-decoration: none;
              color: #28313A;
}

#logo p a:hover { text-decoration: underline; }



/* Page */

#page {
              width: 880px;
              margin: 0 auto;
              background: #FFFFFF;
              border: 10px #FFFFFF solid;
}

/* Content */

#content {
              float: left;
              width: 620px;
              border-right: 1px dashed #DFE1E0;
}

/* Post */

.post {
              padding: 0px 20px;
              margin-bottom: 20px;
}

.post .title {
              margin-bottom: 20px;
              padding-bottom: 5px;
}

.post h1 {
              width: 520px;
              padding: 0px 0 0 0px;
              background: url(images/img08.jpg) no-repeat left top;
              font-size: 24px;
              color: #28313A;
}

.post h2 {
              width: 520px;
              padding: 0px 0 0 0px;
              font-size: 22px;
              color: #28313A;
}

.post .entry {
}

.post .meta {
              padding: 15px 15px 30px 0px;
              font-size: 10px;
}

.post .meta p {
              margin: 0;
              padding-top: 15px;
              line-height: normal;
              color: #28313A;
}

.post .meta .byline {
              float: left;
}

.post .meta .links {
              float: right;
}

.post .meta .more {
              padding: 0 20px 0 18px;
}

.post .meta .comments {
              padding-left: 22px;
}

.post .meta b {
```

```
               display: none;
}


/* Sidebar */

#sidebar {
               float: right;
               width: 230px;
               margin: 0;
               padding: 0;
}

#sidebar ul {
               margin: 0;
               padding: 0;
               list-style: none;
}

#sidebar li {
               margin-bottom: 40px;
}

#sidebar li ul {
}

#sidebar li li {
               margin: 0;
}

#sidebar h2 {
               width: 250px;
               padding: 8px 0 0 0px;
               margin-bottom: 10px;
               background: url(images/img07.jpg) no-repeat left top;
               font-size: 20px;
               color: #28313A;
}

/* Search */

#search {

}
```

```
#search h2 {
              margin-bottom: 20px;
}

#s {
              width: 140px;
              margin-right: 5px;
              padding: 3px;
              border: 1px solid #DFE1E0;
}

#x {
              padding: 3px;
              border: none;
              background: #0A5688;
              text-transform: lowercase;
              font-size: 11px;
              color: #FFFFFF;
}

/* Boxes */

.box1 {
              padding: 20px;
}

.box2 {
              color: #BABABA;
}

.box2 h2 {
              margin-bottom: 15px;
              font-size: 16px;
              color: #FFFFFF;
}

.box2 ul {
              margin: 0;
              padding: 0;
              list-style: none;
}

.box2 a:link, .box2 a:hover, .box2 a:active, .box2 a:visited  {
```

```
              color: #EDEDED;
}

/* Footer */
#footer-wrap {
}

#footer {
              width: 880px;
              margin: 0 auto;
              background: #E5E5E5;
              border: 10px #FFFFFF solid;
}

html>body #footer {
              height: auto;
}

#footer p {
              font-size: 12px;
}

#legal {
              clear: both;
              padding-top: 17px;
              text-align: center;
              color: #595959;
}

#legal a {
              font-weight: normal;
              color: #1B75A9;
}
```

Appendix E: Apache Configuration

E.1 Apache Virtual Host: tutorials

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost

        DocumentRoot /home/caitlinj/website/djcode/

               Alias /site_media/ /home/caitlinj/website/djcode/usersite/media/
               <Directory /home/caitlinj/website/djcode/usersite/media/>
               Order allow,deny
               Options Indexes
               Allow from all
               </Directory>

               Alias /media/ /home/caitlinj/website/djcode/usersite/media/

               <Directory /home/caitlinj/website/djcode/usersite/media/>
               Order allow,deny
               Options Indexes
               Allow from all
               </Directory>

        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>

        <Directory /home/caitlinj/website/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
               allow from all
        </Directory>

        ErrorLog /var/log/apache2/error.log

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel warn

        CustomLog /var/log/apache2/access.log combined

        ##### Django Installation #####

              WSGIScriptAlias /
/home/caitlinj/website/djcode/usersite/apache/django.wsgi

</VirtualHost>

WSGIPythonPath /home/caitlinj/website/djcode/
```

E.2 Django_wsgi.config

```
import os
import sys


path = '~/website/djcode/'

if path not in sys.path:
   sys.path.append(path)

path2 = '/usr/lib/pymodules/python2.6/'
if path2 not in sys.path:
   sys.path.append(path2)


os.environ['DJANGO_SETTINGS_MODULE'] = 'usersite.settings'

import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```