Seventeen Moments in Soviet History

CS4624 - Dr. Edward Fox
Virginia Polytechnic Institute and State University
Blacksburg, VA 24060

Clients:
soviethistory.macalester.edu
James von Geldern
(vongeldern@macalester.edu)
Amy Nelson
(anelson@vt.edu)

Authors:

Karan Bhatia (karanb22)
Zak Edwards (zak279)
Matthew Layser (mlayser)
Austin Moore (aemoore)

Table of Contents

Table of Figures and Table of Tables

Table of Figures

Table of Tables

Executive Summary

The majority of the work done with the website focused on shoring up security flaws and issues. This was first started by identifying all elements of the website that interacted with the database. Next, the corresponding code in the website was found and works was done to correct it. This was done by first identifying how CakePHP handles SQL queries and recommended ways to sanitize SQL queries in CakePHP. Next, flow of control in querying the database was changed to ensure that the recommended changes could be implemented. Once these change were made, the website was first tested to ensure that functionality was not damaged in anyway. Once it was confirmed that the website was still as functional as before the changes, testing was undergone to ensure that the SQL issues were fixed. This was done by attempting to make a SQL injection on the website. The database was then checked to ensure that no changes were made to the website and that the database was in the same state as before the injection attack.

In addition to fixing the security issues associated with the website, general database changes were made as well. First, user registration was changed to ensure that new users were not listed as moderators. Next, all moderators were dropped with the clients representing the only moderators. Then, the website was modified to no longer store passwords in plain text and changed to only store the hashed passwords. This was confirmed by making new users and testing to see if their plaintext passwords were stored. In addition, all plaintext passwords were removed from the database.

Research was also undertaken for notifying the users of the Soviet History website that the website was operational again. First, a script for emailing users was considered but determined to be unacceptable due to a limit on number of emails sent by an address and the fact that the scripts are dependent on the running computer's configurations. Next, a mass email service was considered, but determined to be undesirable as they operator on a monthly subscription fee and the service would only be used once. It was then determined that the best course of action was to determine which users should be emailed and only email them so as to not broadcast to the original hackers that the website was back up.

Finally, work is being done to fix the subtitles not appearing on the audio sections of the website. However, currently they are not working, although, test code is being run to see if it improves the subtitles issue.

1 Requirements

1.1 Description

Seventeen Moments in Soviet History is a digital humanities site for teachers, students, and any other browsers interested in Soviet history. It has been online and hosted by Matrix (MSU) since 2001 and is currently undergoing its second overhaul. The particular objective of this overhaul is to integrate the interactive capabilities of Web 2.0. In particular the goal is to allow registered users to upload and suggest archival documents and multi-media content for inclusion on the site. Content will require the approval of administrators and mid-level managers. Along with this, the site will need to be viewable on mobile devices such as tablets and smart phones.

1.2 Objective

At its present state, the website has many security flaws that are detrimental to both the Soviet History website along with other databases stored on the same server. This represents a hazard to both the users of the Soviet History website and the administrative staff of the servers hosting and maintaining the Soviet History. The objective of this project is to find and fix the security flaws in order to have a running website void of any potential security breaches. Additionally, the website requires implementing authorization of users in order to safely publish content. Finally, the website is riddled with numerous bugs and errors and provided additional time, another objective will be to resolve these issues.

1.3 Document Scope

This document provides a description of the Soviet History website and the associated objectives for working on the website. Next, it provides a description of the users of the website and their day-to-day operations, tasks, and responsibilities (if any). The non-functional and functional requirements are also given and elaborated upon. The design of the system is then further explained with a description of the system architecture, modifications, ues of components, and a data-flow description. Next the user experience is explained and detailed. Finally, the Appendix including Table of Figures with screenshots of the website are given.

1.4 User Audience

Server Administrative Staff

The server administrative staff will ensure that the servers, the website, and corresponding databases are properly maintained and monitored. This includes identifying and resolving any critical issues as well as notifying the website administrative staff of these issues. In addition, should the website be moved to a new location, the server administrative staff is expected to assist in the moving of the website.

Website Administrative Staff

The website administrative staff ensure the upkeep of the website and are directly responsible for any changes or modifications to the code, layout, etc. of the website. They are responsible for reporting any issues or hazards that may affect the server administrative staff to said staff. In addition, the website administrative staff is responsible for making changes to the code and ensuring website quality, safety, and robustness.

Website Moderator Staff

The website moderators help support the adding and removing of content to/from the website. The moderators approve/deny requests to add/remove material and themes to the website as well as enforce the rules of the website.

General Level Users

The general level users are those who will be utilizing the Soviet History website for day-to-day use. They are the ones who will be accessing the site for the material available there as well as requesting material additions/removals. The general level users are the ones who will be most affected by issues with the websites and they are also expected to provide the most feedback in terms of issues with the websites.

1.5 Methodology Used

The source code for the Soviet History website will be studied in order to gain a better understanding of the development efforts of the previous programming firm. This will allow for reduced time spent in debugging the website. After a good understanding is gained, the website will then be edited by the students' development team. The website will be edited to remove bugs found in the PHP source code at this point.

Afterwards, the development team will research proper encryption methods for the passwords. This will allow the development team to encrypt the authentication system to store hashed passwords instead of storing the passwords in plain-text.

1.6 List of Non-Functional Requirements

Safety

Safety is the most important non-functional requirement. This is because the users of the website expect their personal information to be safe and secure from intrusions. In addition, the website should be able to prevent unauthorized access or commands.

Aesthetics

Aesthetics are important because users expect an appealing website. Aesthetics mean the layout, color, and themes of the website match together and aren't garish or clashing. This encompasses ease of use as well. Often, websites which employ old styles and old technology tend to dissuade users from trusting the site and should be avoided at all costs.

1.7 Requirements

This section has been broken into two tables, requirements that should be finished by the end of the semester and requirements that, if there is additional time, will be worked on.

Table 1: Finished Requirements

Name	Description	Method
Local Instances (switched to running on actual server)	Successfully set up local instances for testing purposes	Continue reviewing code and attempting to get an idea of what is occurring and ask for assistance from relevant professors
Database Security	Ensure any calls made to and from the database are secured from any type of attack or SQL injection	Replace deprecated database calls with more up to date SQL interaction methods
Database Cleanup	Purge all passwords from the databases and potentially remove all moderators	Run the appropriate SQL commands
Generate Passwords	Send an email to all users stating they need to create a new password	Set up a script to email users and set-up utilities that find out if it's the first time the user has logged back in.
Streaming Subtitles (Still in progress)	Fix the subtitles so they show up on audio/video	Identify the code written to handle this and properly fix or implement this feature

Table 2: Additional Requirements

Name	Description	Plan of Attack
Tablet Interface	Fix video view on tablet	Figure out how the video is being displayed if in mobile view and adjust the parameters
Smart-Phone Interface	Fix the navigational menu on smart phones to not dominate the entire screen	Figure out how the navigational menu is being displayed on mobile devices and adjust the parameters
Audio/Visual Information	Fix the admin panel to display media filename and corresponding subtitle filename	Find where the panel is set-up and add this to the display list
Image Caption	Have captions appear for the appropriate images	Set up the scripts that display images to also display the corresponding caption
Music Display	Remove unwanted no data available notification	When no data has been found, avoid displaying any additional information
Music Box	Modify music-box to autoplay	Adjust the music-box scripts to play the next available music
Texts and Related Texts	Change links located in texts and related texts to be listed as text	Find where the information for texts and related texts is stored and move appropriate information
Archives	Modify website so hitting archives button displays the archives	Expand the scripts that respond to the archives button being clicked
Subject Bibliography	Adjust add subject function to allow for all information to be submitted	Correct the corresponding scripts
Search Display	Have subject search text put a space between author and text	Find corresponding scripts and update the display
Bibliography	Index rolling search box	Set up scripts to include some type of indexing system
Dashboard	Have the Total Texts and Total Media buttons work	Link the JavaScript functions to the appropriate buttons
Back-End Access Information	Have the information listed for sources on the front-end match with those on the back-end	Clean up the database so the links match
Google Ranking	Include the Google analytic code and meta-tags in the website	Set up the scripts to write the analytic code and meta-tags in the html
Sign-In	Have the databases store how a user is logged in	Fix how the login scripts work for different media types
URL Address	Fix scripts to include new URL address	Find and replace all old URL addresses with the new addresses

2 Design

2.1 Current Design

The website is a multi-page interactive website with profiles which users can use to contribute to the website such as posting comments. A user needs first to create a login and then is able access the website. These credentials are stored in a database. As the user browses, content is delivered to them from another database. Scripts are sent along with this content to allow local interactivity on the web page without having to reload new views, such as opening up a menu. The entire system is based on a Model-View-Controller, or MVC, architecture which is described in more detail below.

The website is written in PHP: Hypertext Preprocessor (Or, PHP for short). PHP allows the user to deliver content onto web pages dynamically. The Cake-PHP library is used to hold a MVC framework for the project. For the backend server, a MySQL server is used to hold the information in a database.

The Cake-PHP library allows the data to be pulled in from the MySQL database, and displays it in an efficient manner on the webpage.

2.2 Components Used

The following components are used in the building of this application:

Apache Tomcat Server:

An HTTP server used to host the website. The Tomcat server allows us to deliver dynamically generated HTTP webpages to the client. The web server handles requests to/from a port, which then is sent to the client via port forwarding from a Network Address Translation (NAT).

MySQL Server:

A server holding a MySQL database. The MySQL database stores information that is used in the application. The server then handles the request to and from the server and routes that information back to the web application. This supports inserting and deleting from the database, as well as querying the database for entries.

PHP:

Cake-PHP: An open source framework for building web applications. It adopts the Model-View-Controller (MVC) architecture and is written in PHP. With very similar functionality to Ruby on Rails, CakePHP stresses popular software engineering principles and design patterns, such as Front Controller, Association Data Mapping, and Active Record.

Fire-PHP: A PHP debugging add-on for Firefox. This add-on allows us to find where the bugs existed in the codebase. Finding where the bugs were allows us to be able to fix them at a faster pace. This shortened our debug time significantly.

PHPMyAdmin: free software tool written in PHP that handles the administration of MySQL over the Web. This supports a wide gamut of operations within MySQL, which is the database used in this project. Frequently used operations (tables, managing databases, relations, users, permissions) are performed via the user interface, while still allowing a command line option to directly execute any SQL statement.

JavaScript:

Bootstrap.js: A front-end framework for JavaScript. This framework allows for easier customization options for the User-Interface. Bootstrap gives the front-end developer a wide array of options and objects to use that have already been customized. The Bootstrap library is written in both JavaScript and CSS.

JQuery.js: JQuery is a special JavaScript library that helps reduce the amount of code necessary to write for various JavaScript activities. It helps provide abstractions for low-level features such as Document Object Model (DOM) manipulation, document navigation, animation creation, AJAX calls, and other event handling.

livevalidation.js: Special JavaScript library for creating interactive validation forms. This helps give notifications to the users about incorrect fields in their forms before they click the submit button.

Charisma: Special JavaScript library for adding themes to an HTML5 template. This helps easily add several pre-configured themes to a website within a minimal amount of time and with less written code.

excanvas.js: Special JavaScript library that adds HTML5 canvas functionality to older browsers that don't initially support it. This is necessary as Charisma and other HTML5 templates in the website use the canvas tags.

binaryajax.js: Allows AJAX calls to be made using binary data and receive binary data from the source. This allows for faster upload and download of video, audio, and image sources.

id3.js: JavaScript library for reading and modifying id tags in MP3 files. This allows for posting information about audio on the website as well as modifying downloaded audio files for better accuracy.

audioplayer.js: JavaScript library that works with JQuery to add an audioplayer tool at the bottom of audio files played on the website. It includes a play/pause button, a scroller and a volume control tool.

contentslider.js: JavaScript library that works with JQuery to add an automated resizing tool. This allows the website to changes its layout and display whenever the browser size changes. In addition, this helps provide mobile browser support.

Foundation Responsive Library: Special library that includes additional features for a more “responsive” website. This allows the website to be more easily read and easier to navigate. It provides additional data and media options than those provided by JQuery. In addition, it helps speed up some JQuery requests.

Linkedin API: JavaScript API that allows for Linkedin to be supported on the website and allows users to login and/or register an account with their Linkedin profile.

Facebook API: JavaScript API that allows for Facebook to be supported on the website. In addition, users can login and/or register an account using their Facebook profile.

Other:

MySQLAdmin: Special GUI for viewing databases, tables, etc. in a MySQL server. This allows users to make commands and view information without inputting commands into the terminal. In addition, it makes database management and manipulation faster and easier.

2.3 System Architecture

The website is hosted on a test server and is planned to be merged onto a full-time web server when all of the security issues and flaws have been fixed. This website is the front-end for the Soviet History application that displays information about

Soviet-era artifacts. The website hosts videos and images about events that occurred in the Soviet Union. In addition, it features a large encyclopedia about various events and facts about and in the Soviet Union. Users can request to post additional links and information to be added to the website by moderators and admins. Users can register to make these requests by either creating an account or signing in with their Facebook or Linkedin accounts.

In addition, teachers and instructors can create a syllabus corresponding to material hosted on the website which allows them to include information, questions, and links to a syllabus for use in their class.

Some of the changes to be made to the website include removing SQL security flaws and any other known security defects. In addition, a tiered hierarchy of user privileges is going to be set up as well to provide additional security padding between the user and the actual code. Finally, a list of known bugs and issues will also be fixed, but this was determined not to pose any necessary significant changes to the website design.

The MySQL server hosts the backend database. The MySQL server is kept because it was decided to be the easiest to use and maintain. In addition, the website is coupled to the web server and would provide an unnecessary amount of working in moving to a different database management technique.

Apache Tomcat is used to host the website on a local server. This was chosen as the server running the website is an Apache Server and hosts all websites using Tomcat.

2.4 Data Flow

The data flow is that of a standard MVC framework. The user makes some kind of request on a page located on the website. The appropriate controllers for those requests then process the input so the models can handle the information and make the appropriate calls. The corresponding models then fetch the data or in some way update data stored on the backend and send the status or requested data back to the controllers. The controllers then update the page view appropriately for the user.

2.5 Schema/DataTables

The server stores information about:

- User Accounts - this stores information regarding user accounts, such as name, date created, user_id, etc.
- Images - this is information regarding images hosted on the website, such as image_url, image_size, etc.

- Videos - this is information regarding videos hosted on the website, such as video_url, video_size, etc.
- Links to audio locations - this is information regarding the location of audio files, such as audio_url, audio_size, etc.
- Sources and links - this is information regarding sources and links to other websites. This has attributes of source_url, link_url, date_created, etc.
- A glossary of terms and specific information related to the posted information
- Content requests - Requests made by users for adding additional information about a specific topic
- Available subjects - Relates to the glossary in that it stores a list of what topics can be viewed

2.6 User Experience

The User Experience for the website is aimed at being aesthetically pleasing. A majority of how the user interacts with the given website is already completed. As a result, this project is to just finish any bugs/issues that hinder the User Experience. The User Interface will be a clean layout that will involve a list of images and videos. Additionally, the end-user will be able to login to the website in order to upload videos and images, as shown in Figure 3.

3 Implementation

3.1 Minimization of Issues/Errors While Working

Since work will be done on the same server that the website is operating on, care must be taken to help minimize the impacts of introduced bugs and errors. First, any file that will be modified will first be backed up so, in the event of an issue, the old file can be put back into use and help restore the website back to working conditions. In addition, a copy of the original file will be made and all changes and modifications will be done to that file. When the file is ready to be tested and/or used, the test file will replace the actual file. Following this, the website will be extensively checked for any introduced bugs/errors that the new test file may have caused. If any are found, the backup file will immediately replace the test file and the test file will be debugged and modified to remove the offending behavior. In this way, it should be expected that any bugs/errors created during the implementation process be as minimized as possible.

Because the codebase is held on the server the already-existent website is operating on, no code-repository is used. This is because the code-base is not hosted on any code-repository server. This makes the task of collaborating with multiple developers on our team more difficult because there could be multiple team members working on the project at once. That is why it's expected that group members check the recently modified time and the existence of swap files to help see if another group member is working on the same file. If so, the individual should either work on another file, or make another test copy and work on that file instead. In addition, the individual should let the other group member know so any changes made can be merged together.

Additionally, the development team has a group chat where we communicate about the efforts on this project. This is useful because we mention when we will start our work, what work will be done in that session, as well as when we are done. With this, our team is able to work better and communicate effectively.

3.2 Division of Labor

Due to the nature of modifying a website with a large amount of legacy code, everyone, for the most part, will be coding. Since there exist a lot of bugs/issues that

are reported for the website which cannot be grouped into different categories, there's no way to efficiently divide tasks. Instead, group members will pick tasks they wish to work on or a task that is related to a file that no one is currently working on. In addition, group members are expected to collaborate on tasks and pass tasks around if an issue comes up that they themselves cannot solve.

However, there exist several unique tasks that are assigned to each group member. These tasks are as follows:

- Zak Edwards is to be the primary liaison between the clients and the team.
- Karan Bhatia is to be the primary editor of any deliverables and authorize any edits or changes to the deliverables.
- Matthew Layser will help ensure that the team is sticking to the schedule predefined and that the team will submit deliverables on time.
- Austin Moore is to help ensure that, in the event of a server change, the code can be easily migrated. In addition, because of his prior experience in web-design, he is to be the primary contact for any coding issues if they arise locally to our team.

3.3 Development Tools

As the majority of the programming will be done on the actual server the website is hosted at, SSH and a number of SSH clients will be heavily used. For instance, the Secure Shell Chrome extension will be used. In addition, SFTP clients may also be used to copy files to and from the server if a group member's environment is more set up for that.

In terms of actual coding, a large number of text editors will also be used. For example, Vim, Vi, Emacs, and Nano. In addition, Microsoft's Visual Studio IDE was also used and several plugins for allowing for SFTP transfers have been used as well.

3.4 Timeline

Below is a list of the requirements table modified to have an estimated time completion and a justification for why the team feels it will take that long to complete a task. Although the team will make full efforts to meet time requirements, due to unforeseen circumstances, tasks may take longer to complete, or may actually be completed earlier than anticipated.

Table 3: Finished Requirements (Timeline)

Name	Estimated Time Completion	Justification
Local Instances (Switched to running off local instance)	2-3 Weeks	First the team has to be able to gain access to the most up-to-date website code and databases. Then, the team must download the appropriate software and setup the software correctly. Finally, the code must be modified heavily to be able to run off the local instance as there were numerous hard-coded sections
Database Security	3-4 Weeks	This is most important as the website's security is the most paramount concern. Any SQL queries have to be sanitized and passwords must be encrypted. In addition, the site must be thoroughly tested to ensure security
Database Cleanup	2-3 days	Simply dropping the appropriate tables and ensuring the website didn't need them
Generate Passwords	1-1.5 Weeks	Requires finding appropriate mailing component for large-scale emails. In addition, requires setting up a script to pull email addresses from databases and email users.
Streaming Subtitles (Still in progress	2-3 Weeks	Requires intimate understanding of the audio/video third-party libraries and ensuring the

		databases actually contain the subtitles.

Table 4: Additional Requirements (Timeline)

Name	Time Completion	Justification
Tablet Interface	Rough Estimate 1-2 Weeks	Requires intimate understanding of the mobile visualization library and correctly testing on a wide range of mobile devices
Smart-Phone Interface	Rough Estimate 1-2 Weeks	Same as for Tablet Interface
Audio/Visual Information	Rough Estimate 1 Week	Requires finding the appropriate panel and modifying it
Image Caption	Rough Estimate 1 Week	Requires finding the scripts to display images and setting them up to include captions
Music Display	Rough Estimate 2-3 Days	Requires putting conditional statements in the code
Music Box	Rough Estimate 1 Week	Requires intimate understanding of the audio player library and appropriately modifying them
Texts and Related Texts	Rough Estimate 2-3 Days	Should just require modifying the databases and setting up a script to properly do that
Archives	Not Estimated 2-3 Days	Requires implementing the archives button to actually have an associated event
Subject	Not Estimated 1 Week	Requires debugging and tracking down the offending database modification code and correcting the issue
Search Display	Not Estimated 1 Day	Requires putting a space in the output display
Bibliography	Not Estimated 2-3 Weeks	Requires building appropriate indexing suite or finding and implementing an already existing suite
Dashboard	Not Estimated 1 Week	Requires setting up appropriate handlers for the buttons
Back-End Access Information	Not Estimated 1 Week	Requires searching through the databases finding any discrepancies between what the website displays and what is stored in the database and correcting any issues
Google Ranking	Not Estimated 1-2 Days	Requires including the tags into the templates
Sign-In	Not Estimated 3-4 Days	Requires modifying how the user logs-in

URL Address	Not Estimated 1 Day	Requires a search-and-replace and ensuring the site is still functional

4 Prototyping

4.1 Prototyping

As the majority of requested information has been discussed in other sections, this will simply be an update on what specifically has been accomplished. Due to the secure nature of the work, screenshots and other kinds of images could not be taking due to it being a privacy concern and security leak.

So far, the majority of the work accomplished has been security oriented. All calls and queries to the database has been updated to be sanitized and secured. This was accomplished by researching how CakePHP queries a SQL database and fixing inappropriate queries or changing method sections to allow for a more correct way of querying the database as suggested by CakePHP documentation. As a result, malicious users are unable to insert SQL statements into any text-field and have their query be run by the database. This is a huge accomplishment as a reason the website had an attack at the beginning of the semester was because of the ability to execute SQL statements from the GUI text boxes.

Next, the way users are registered and how users log in was changed. Originally any new user was automatically considered a moderator. This issue was rectified and now any time a user registers for an account, they are labeled as only a user, not a moderator. In addition, the unhashed version of the user's password was stored in the database. This has now been changed so that it is no longer the case. In addition, the password is now hashed using md5 and that hashed password is stored in the database. All plaintext passwords were purged from the database. This is important because the passwords being stored in plaintext were a significant security issue. Anybody who had access to the SQL server (Whether the access was authorized or unauthorized) would have been able to see the passwords of every user. Now, because the passwords are stored as an MD5 hashed string, users who have access to the SQL server will be unable to see the passwords of any user.

Currently, the work is focused on penetration testing, scripting, and cleaning the SQL databases. The SQL database contains a large number of unused databases, so those were dropped, in addition, all moderators were dropped and the appropriate users were re-added as moderators. In addition, penetration testing is being performed to help ensure the website is actually protected against from SQL injections. Finally, a script is being created to inform users to update their password for the account and a high-volume mailing client will be found in order to send these emails.

5 Refinement

5.1 Refinement

As stated before, the majority of the work is being done on the security of the website. and the server administrator for the website has been contacted. We asked him to run his own suite of tests on the website to confirm that the website is secure. We received minor feedback that needs to be incorporated into the code before he signs off on us putting the site up again.

The script to send users emails is still under development. The site needs an email server to send the password reset emails from, so that the emails are from a credible source, and not a personal email address. We met with one of our clients and were told that this feature may be scrapped in favor of a private announcement. We will simply flush all the existing user accounts and have the users make new accounts. This is to avoid the possibility of sending the email to the hacker, notifying him that the site is up.

In addition, we are actively working to understand the cause and solution of the lack of subtitles for the videos on the site.

We have received feedback on what security issues there are with the website. Although all SQL injection risks appear to have been removed, there is a risk with xsrf exploits.

6 Testing

6.1 Testing

The majority of the testing that was done was focused on SQL injection testing. This means we attempted to write SQL injections on the website in an attempt to manipulate the database. We knew that the SQL injection was a success if there was a newly added record on the database. To do this we examined all areas where the code was changed to make it SQL safe. Following this, how the code affected the website and what aspects the code represented was identified so that a thorough testing would be undertaken. Next, all associated website elements: searching, login, user registering, etc., were tested for SQL injections. This was done both before the code change and after to ensure that the update actually fixed the issue. If it was found that the element was still vulnerable to a SQL injection, the code was reviewed again for any potential database queries and modifications that were missed. This code was then modified to be properly SQL safe. In addition, the site was also tested against URL SQL injections to ensure safety in that regard as well.

In addition to SQL testing, the website was generally tested for functionality whenever the code was changed as well. This means that anytime a script was updated, the website would be tested extensively to ensure that the change didn't break functionality. If it was found to have broken functionality, the change was removed and modified to ensure that it would no longer break functionality. This is to make sure that working elements of the website would remain that way.

Finally, the Michigan State University server administrator, Dennis Boone, was contacted to perform additional security tests on the website. This was to ensure that any potential security leaks would be found and fixed before the website went live. He confirmed that we have met the requirements of protecting the website from SQL injection exploits.

The testing results were left out due to security concerns as they are just direct SQL injection tests. In addition, they reveal critical information about the website's database and inner structure.

Lessons Learned

The schedule essentially went as the first few weeks being spent attempting to get a mock host of the website up and running and ultimately failing to do so. As such the team switched to working directly on the server the website is hosted on and immediately began code development. This started off slowly as the code was largely uncommented and there are a large number of files associated with the website. This meant the team had to read a large amount of code in order to understand where and how they should make any changes. This process took about two weeks.

Next, development began on fixing security flaws. This essentially revolved around fixing the SQL issues associated with the website. This coupled with the testing took up the majority of the semester. Finally, modifying certain database issues and moderator issues, as well as research on the emailing issues occurred while the security flaws were being addressed.

One of the biggest problems was getting up and running with all of the code. The team originally attempted to host a mock version of the website for testing purposes. However, with how hard-coded the scripts were and how much software setup and configuration needed, this process would have taken much more time than available. As such, the team had to test on the server the website is hosted on. This meant the team had to be much more careful about making changes, as it affected actual usage of the website. To help prevent accidentally breaking of functionality, the team made backups of every script they modified and if functionality was found to have been broken, the original scripts were put back up until the new changes correctly worked with the website.

Another issue was learning CakePHP, as none of the team members had any experience with this framework. As such, time was spent reading and understanding the CakePHP documentation so as to be understand how to improve and change the website without damaging functionality.

Some of the future work will involve fixing bugs and errors reported in the tables found in this document that the team was unable to solve as well as any other discovered issues. In addition, some work should be undertaken to better improve the readability of the code as the lack of comments was a major factor in how long it took to make any changes.

Acknowledgments

Dr. Edward Fox - Solving initial issues and answering queries about documents
Dr. Amy Nelson - Providing assistance with communication, providing guidance on the website
Dr. James von Geldern - Providing guidance on the website, answering queries
Dr. Lewis Siegelbaum - Providing guidance on the website, answering queries
Dr. Kristen Edwards - Providing guidance on the website
Mr. Dennis Boone - Running security test suite on website, and answering technical queries

Appendix

Table of Figures

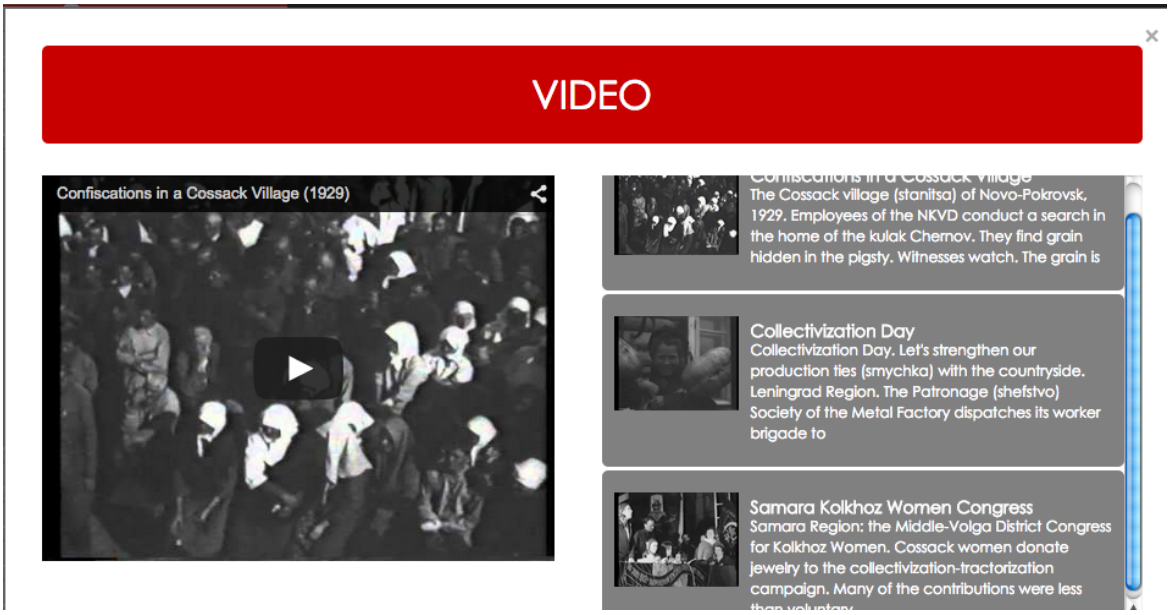

Fig. 1 A screenshot of dialog displaying videos related to a particular subject.

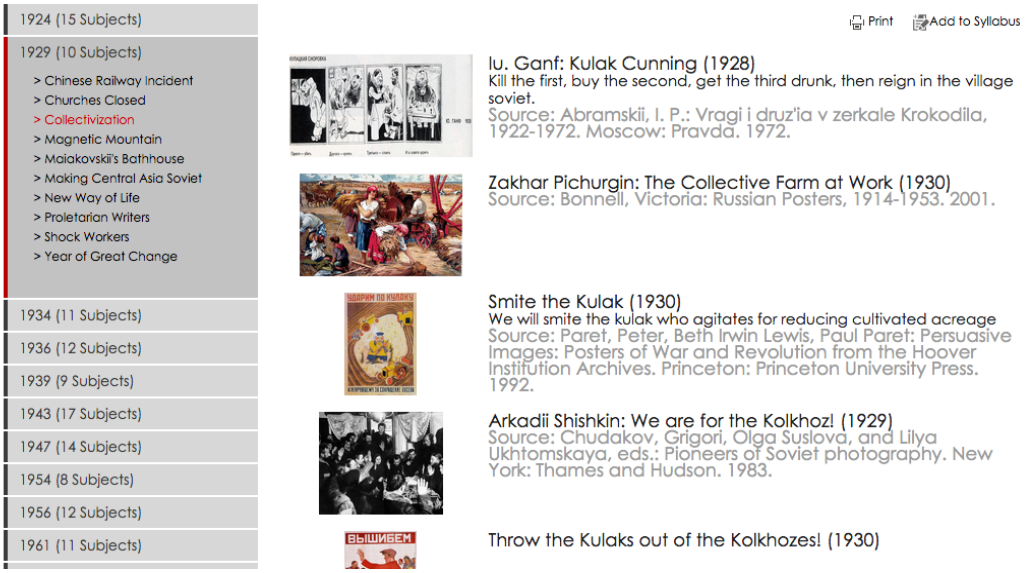

Fig. 2 Screenshot of list of aggregated content for a time period (1929).

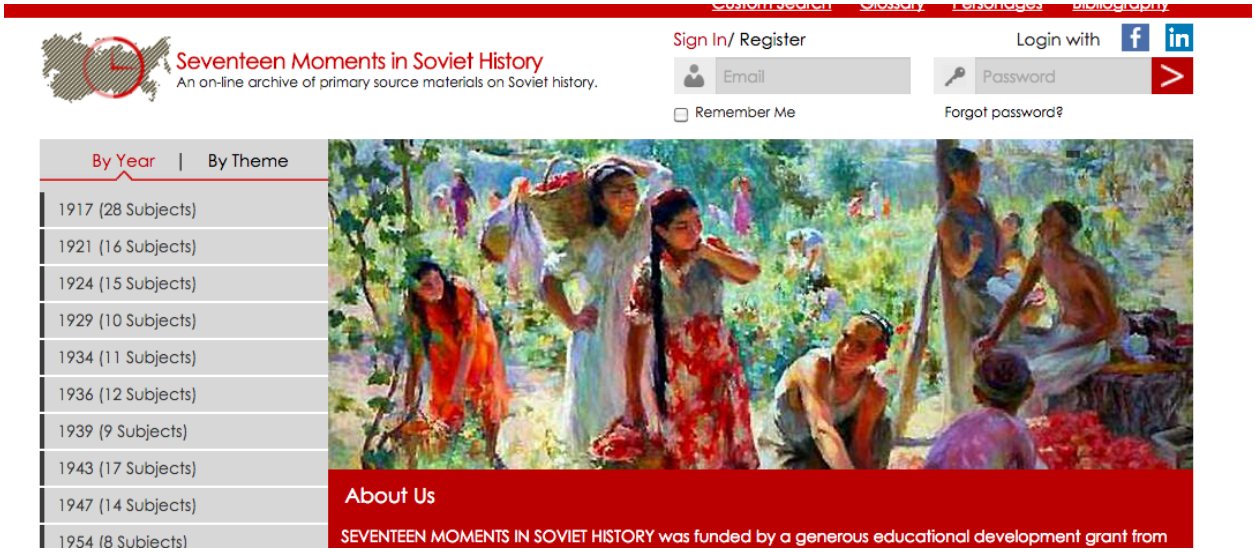

Fig. 3 Screenshot of home page with login and side navigation bars