RENOWN

Purpose

The purpose of this document is to provide evidence for the different roles of the team members.

Introduction

Team Renown analyzed, designed, implemented and tested a web application to meet the requirements of the CS3733 Software Engineering Star Trek Mashup Project. This application, similar to iMovie, provides users with the capability to construct larger videos from video segments. Like most video-parsing applications, the Star Trek Mashup Project can create playlists, upload local video segments, as well as grab other video segments from remote site libraries. The overall function of this application exists to prove that our team is capable of constructing a web application that implements Amazon Web Services, JavaScript, RDS Database, Java code, API Design, and HTML.

Team organization, members, and responsibilities

Our team had a loose organization without defined roles or meetings. We scheduled three regular meetings per week (Tuesday 5-7, Thursday 5-7, and Friday 6-9) as well as scheduling additional work time as needed.

- Nathan Jackson - wrote javascript, tested and implemented lambda functions
- Thomas Vose - wrote tests, managed API
- Sean Morrissey - wrote javascript and implemented html.
- Eric Reardon - wrote lambda functions, managed database, java implementation

Process

We started by identifying all of the features we would need in our application, and then designing our UI to fit. We first made the HTML and CSS to create the user and admin interfaces. Next, we wrote lambda functions and uploaded them to AWS and wrote a simple test HTML and javascript file to test the basics of each of the functions. After we confirmed that the lambda functions were behaving as expected, we implemented them into the final versions of the UI. After adding all of the intended features, we wrote test cases that would verify that our features worked as intended anytime code was changed.

Our work process consisted of starting programming at our scheduled meetings and then assigning code to work on before the next meeting. Once we met again we would collaborate on issues we faced as well as moving onto the next set of features we needed to work on. As the term went on we discovered that this often left us working long shifts the night that an iteration was due, so we decided to start meeting earlier in the week so that we had more time to resolve difficulties.

Communication was done through a group text and a when2meet was used to schedule meetings. Outlook events were then used to keep track of these meetings and make them official.

Tools

AWS was used to store all of our segments as well as the database. The database was managed using mySQL workbench, and then stored as an instance in AWS RDS.. All java coding was done using eclipse enterprise edition, and initially code was shared using GitHub, however, we eventually ran into issues with merging and other various errors. We kept the project on GitHub but switched to using GitKraken to manage cloning and pushing. Brackets was the IDE we used for both javascript and HTML coding. When we had a CORS issue, all testing had to be done by uploading the HTML and .js files to AWS and running it with an extension but after the CORS issue was resolved the live previews in Brackets made the design go much smoother.

Accomplishments

Use Cases:

Create Playlist	
Participating actor:	Initiated By Participant
Entry Condition:	None
Exit Condition:	Empty playlist screen is shown and empty playlist is created in the library
Flow of Events:	1. Participant requests to Create Playlist 2. Renown Web App responds to request by creating an empty playlist and updates display
Show Library Playlists	
Participating actor:	Initiated By Participant
Entry Condition:	None
Exit Condition:	Library Playlists are shown in playlist window
Flow of Events:	1. Participant requests to Show Playlist 2.Renown Web App responds to request by showing playlist window and updates display
Delete Playlist	

Participating actor:	Initiated By Participant
Entry Condition:	Playlist exists
Exit Condition:	Playlist is removed from library
Flow of Events:	1. Participant requests to Delete Playlist 2. Renown Web App responds to request by removing the playlist from the library and refreshing display
Upload Segment	
Participating actor:	Initiated By Participant
Entry Condition:	None
Exit Condition:	Video Segment is shown in library tab window
Flow of Events:	1. Participant requests to Upload Segment 2. Renown Web App responds by opening file explorer prompt 3. Participant navigates to file locally and opens segment 4. Renown App displays new segment in library tab and refreshes display
Delete Segment	
Participating actor:	Initiated by Admin or Participant
Entry Condition:	Segment Exists
Exit Condition:	Segment is removed from Library
Flow of Events:	1. Admin or Participant requests to Delete Segment 2. Renown Web App responds to request by removing the segment from the library and refreshing display
Append Segment	
Participating actor:	Initiated By Participant
Entry Condition:	The playlist exists, is open, and the segment exists in the library
Exit Condition:	Selected segment is appended to the current playlist

Flow of Events:	1. Participant requests to Append Segment 2. Renown Web App responds to request by adding segment to end of playlist and showing it in the current playlist timeline and refreshing display
Remove Segment	
Participating actor:	Initiated By Participant
Entry Condition:	The playlist exists, is open, and the segment exists in the playlist
Exit Condition:	Selected segment is removed from the current playlist
Flow of Events:	1. Participant requests to Remove Segment 2. Renown Web App responds to request by removing the segment from the opened playlist and refreshing display
Show Playlist Segments	
Participating actor:	Initiated By Participant
Entry Condition:	Playlist exists
Exit Condition:	Segments in current playlist are shown in the timeline window of the selected playlist
Flow of Events:	1. Participant requests to Show Playlist Segments 2. Renown Web App responds to request and shows the segments in order that exists in the playlist and refreshes display
Show Local Segments	
Participating actor:	Initiated By Participant
Entry Condition:	None
Exit Condition:	Local video segments are shown in the segments tab window
Flow of Events:	1. Participant requests to Show Local Segments 2. Renown Web App responds to request by showing the local video segments in the segments tab window and refreshes display
Search Segments	
Participating actor:	Initiated By Participant
Entry Condition:	None
Exit Condition:	Local and remote video segments containing search criteria are shown in

	segments tab window
Flow of Events:	1. Participant requests to Search Segments 2. Renown Web App responds to request by matching search criteria to local or remote segments and updates display with those that match
Play Playlist	
Participating actor:	Initiated By Participant
Entry Condition:	Segments exist in the selected playlist
Exit Condition:	Video segments are played in order from timeline on the playlist play window
Flow of Events:	1. Participant requests to Play Playlist 2. Renown Web App responds to request by playing individual segments in order from timeline
Mark Segment	
Participating actor:	Initiated By Admin
Entry Condition:	Local segment exists
Exit Condition:	Selected Local segment is marked for no remote use
Flow of Events:	1. Admin requests to Mark Segment 2. Renown Web App responds to request by making selected segment marked and unavailable for remote use
Unmark Segment	
Participating actor:	Initiated By Admin
Entry Condition:	Local segment exists
Exit Condition:	Selected local segment is unmarked to allow for remote use
Flow of Events:	1. Admin requests to Unmark Segment 2. Renown Web App responds to request by making selected local video segment unmarked, so it is now available for remote use
Register Site	
Participating actor:	Initiated By Admin
Entry Condition:	Another remote Library exists

Exit Condition:	Another remote Library is imported and remotely available for current site
Flow of Events:	1. Admin requests to Register Site 2. Renown Web App responds to request by adding segments from remote library to segments tab window and updating display

Features not yet implemented:

The only feature that we did not have time to implement is the ability to acquire remote segments into the library. We are able to register and unregister a remote site, but it does not add the remote segments to the library. Also, this means the mark segment functionality is obsolete, but it does work and edits the 'availableRemote' column in the database.

Additional Features:

Although it was not required, we tried to make a more professional looking User Interface that would (in the long run) be more intuitive and functional in a video compilation application.

Deliverables

Entire Java Development Environment	Our Eclipse workspace holding our Java code and JUnit tests.
README Manual	A set of instructions detailing our project and exactly how to use our landing pages.
Code Coverage	Ecl Emma code coverage of JUnit test cases.
Git Log	A generated log of commits showing all that was done and communicated through Github.
Landing Page Links	Admin landing page: https://renownsegments.s3.us-east-2.amazonaws.com/admin.html User landing page: https://renownsegments.s3.us-east-2.amazonaws.com/user.html

Reflection

What worked, what didn't work

Trying to rely on our meeting schedule to be enough to get an assignment done didn't work out. We learned that on top of our scheduled meetings we had to improvise some and do meetings at any time available throughout the week, especially earlier so that we could frontload the work on each iteration. This issue left us a little bit behind after the first iteration, but we were easily able to catch up for the second.

Our biggest mistake

As a team, our mistakes did not come from how we worked together, but more from the skills that we brought to the table and how we used them. For example, a large part of our issues stemmed from the way we handled our JavaScript in the web application. All of the code necessary for our web app to perform was in one file. Because of this, a large part of our webpage's functionality was very spotty at certain points, most of which we ended up repairing by the final iteration. Due to our lack of knowledge in web development as a group, we were unsure how to handle multiple js files and the web app's backend as a whole. To avoid this mistake, it would have been best to split this file into multiple of which handle different portions of the web page. It would also be good to figure out how these multiple js files would interact with each other to update the web page.

Changes we would make

To avoid the problems we had during the process, it would have been a good idea to come up with a simpler UI design to handle the web app. We agreed upon something that seemed to be viable for the 7 week term, but due to our lack of knowledge of UI design, we soon found out that what we planned for was ambitious to say the least.

It would have been a good idea for us to try to learn more about web development as a group within the first week or two. We could have been better prepared to create an even more functional project because of it. It would have also allowed us to better understand the logical errors that occured within the interactions between our javascript and the S3 bucket.

Lessons learned

There's 2 things you don't mess with: Texas and JavaScript. -Professor Heineman
We learned to make sure our API has everything you could possibly think of before you upload it, because otherwise you'll have to reupload it and reconnect all of the lambda functions which takes a lot more time than just being all-inclusive the first time you upload it.

Advice to future teams

The biggest advice we could give is to make sure you know HTML and CSS before you take this class or at least be familiar with it. At the time the first iteration was due Sean was the only one who knew HTML so this was a severe bottleneck and put a large workload on him. Additionally we recommend meeting at least once right at the start of an iteration so that you can divide up work and get an idea of how much work might be there. We started late on our first iteration because we assumed there would be very little work to do which resulted in us working all day leading up to the due date and still having issues.