Video Discussion Board System

Architectural Document

Team Members

- Francis Lagueu Fogang
- Abhishek Mallepalli
- Valera Maniuk
- Mikael Mendoza
- Jordan Yelloz

Features

The user-facing component of this system will be contained entirely within a web browser. All interactions between the user and the system will take place with a web application.

Video Upload

Topic

The system should provide users the ability to upload a video file meeting the video file requirements supported by the system as a new Topic of discussion.

Response

The system should provide users the ability to upload a video file meeting the video file requirements supported by the system as a response to a Topic or other Response. Each response can be submitted at a specified temporal position within the Video associated selected Topic or Response.

Video Playback

Topics

The system should provide users the ability to watch video topics. During the playback of video topics, users should be allowed to view responses to the video that are near the current playhead and offered a summary of the available responses at any point within the video.

Responses

At any point within a video, whether a Topic or Response, the user should be allowed to submit a Response.

Performance Metrics

The Video Discussion Board System contains a few bandwidth-expensive features, specifically network transfer of video files. The transfer time (upload or download) of video files is constrained almost entirely by environmental conditions, specifically the minimum transfer speed of the link between the video server and web client. Additionally, the playback of video files is affected by the latency of the link between the server and client. The problem is amplified when attempting to seek a video. The principal metric that is important to this project is the latency from the selection of a content item for playback to its eventual playback. This metric will be termed **seek latency**. This project will attempt to minimize this latency for any specified video bitrate to a value that is considered acceptable to the user.

Software Development Practices

Coding Conventions

- 2-space indentation
- 80-column line width
- Class Naming
    - `CamelCase`
- Function/Method Naming
    - `lower_case () { }`
- Variable Naming
    - `lower_case`

Toolset

Client

Programming Languages

- JavaScript, CSS, HTML5

User Interface Framework

- Polymer

Video Playback

- Video.js with DASH plugin

Server

Programming Languages

- Python

Web Service

- Django

Data Persistence

- Django

Video Transcoding and Thumbnailing

- Amazon Elastic Transcoder

Large Content Storage

- Amazon S3

Interfaces

The client application will employ a few core interfaces when dealing with the video content. First, each Media Content item will be represented with a metadata object. Response content items contain additional information. Content metadata objects will be loaded from a Media Catalog. Content assets (actual video, audio, and images) will be resolved with a Media Repository, a simple tool which maps content identifiers to URIs. These domain interfaces should provide the necessary paths for navigating the network of content stored in the system and retrieving the assets for presentation.

MediaRepository

The MediaRepository interface provides facilities to supply the location of video content when given its content identifier. The video player queries implementations of this interface when loading video content.

Methods

- resolve_manifest(id: String): string
  - Given a content identifier, supplies location (URI) of video data.

- resolve_poster(id: string): string
    - Given an content identifier, supplies location (URI) of video poster image.

MediaContent

The MediaContent interface is a generic interface for all video content that is playable by the system. This is simply a metadata structure for the player to process.

Properties

- content_id: string
- duration: number
- title: string
- author: string

MediaContentResponse

The MediaContentResponse is a subclass of the MediaContent interface which contains additional metadata fields necessary for Media Content that is a response to another Content item.

Properties

- timestamp: number
- parent_content_id: string

MediaCatalog

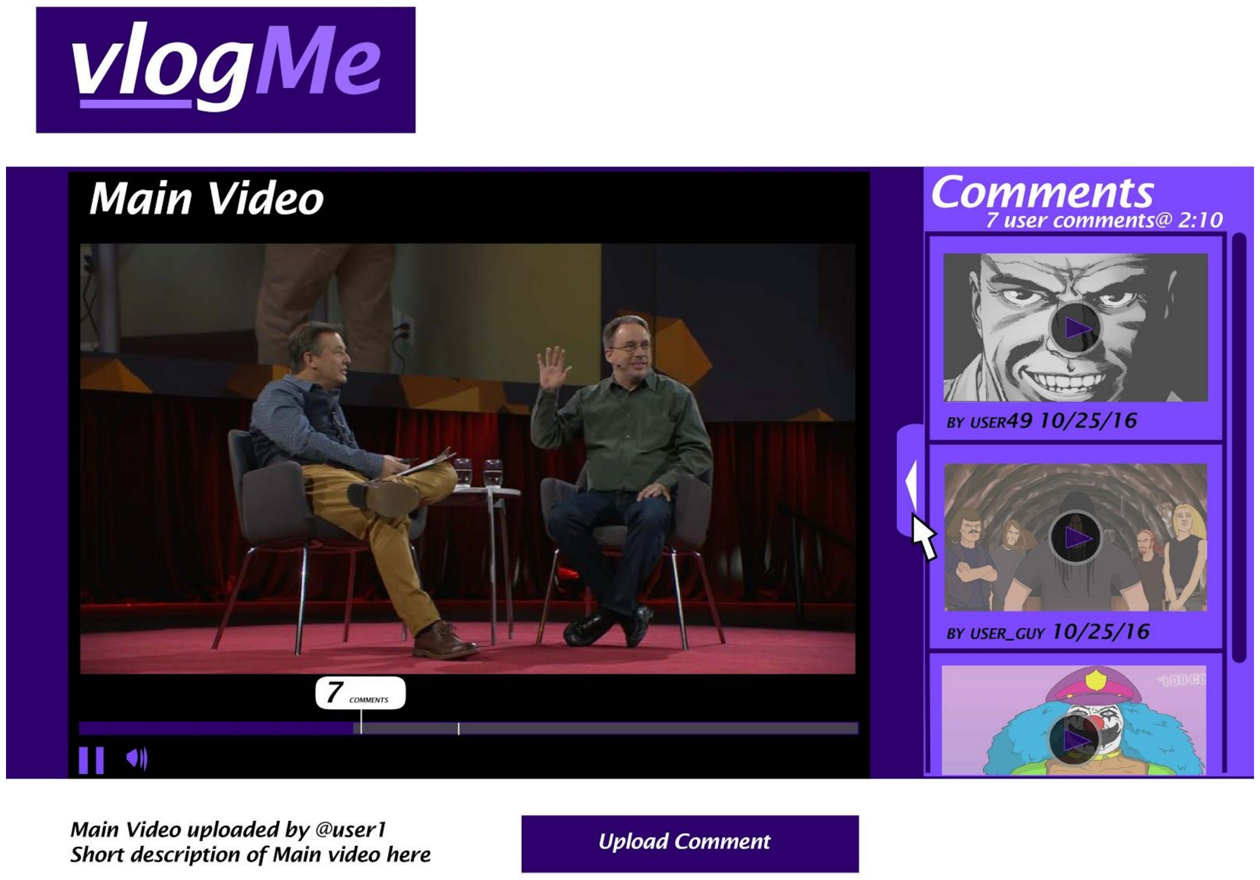
The MediaCatalog interface is a data source which is queried by the client application for MediaContent items. Implementations will typically be backed by a web service.

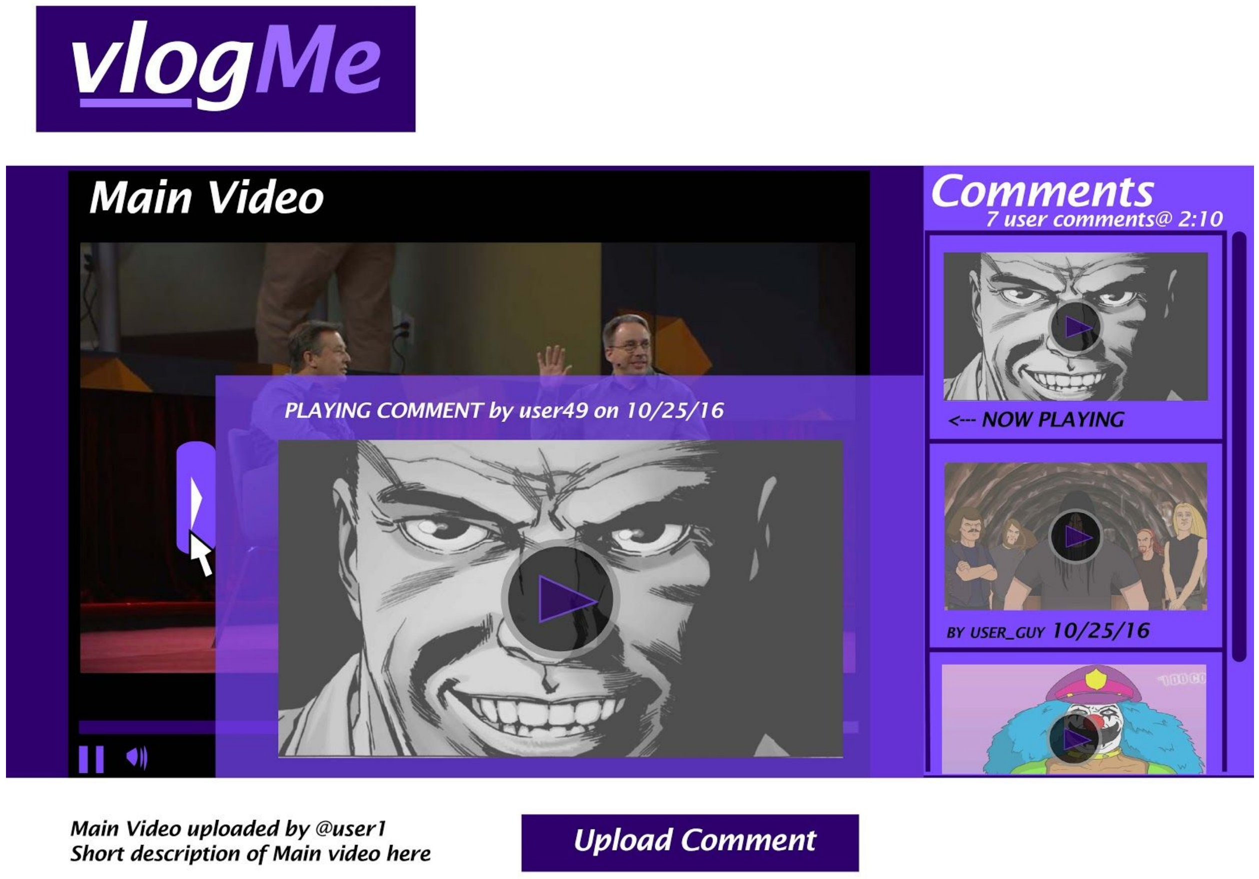
Methods

- latest(limit: number, offset: number): Array<MediaContent>
- responses(id: string, limit: number, offset: number): Array<MediaContentResponse>
- responses_count(id: string): number
- topics_by_user(user_id: string, limit: number, offset: number): Array<MediaContent>
- topics_by_user_count(user_id: string): number
- responses_by_user(user_id: string, limit: number, offset: number): Array<MediaContentResponse>
- responses_by_user_count(user_id: string): number

User Interface Design Concept

- Below is a sample Layout for the project, the Main Video frame represents the video player for the main post in a video blog thread.
- The Comments panel to the right displays video responses by users for the nearest point in the Main Video containing responses.
- The Comments panel can dynamically expand to play the top responses, sliding from left to center, over the Main Video frame.
- The Main Video is always accessible, and the Comments panel updates to display the appropriate responses according to the point in the Main Video that is playing.




vlogMe
Main Video
Comments
7 user comments@ 2:10
PLAYING COMMENT by user49 on 10/25/16
<--- NOW PLAYING
BY USER_GUY 10/25/16
Main Video uploaded by @user1
Short description of Main video here
Upload Comment

Necessary Additional Features

The following are features necessary for the usability of the final submission:

- Interface elements allowing users to add parent video submissions.
- Interface elements allowing users to add response video submissions.
- Response videos must play directly to the right of the textual link submitted with the video. This allows the eyes to see the response video while moving very little from the parent video and response links. An attractive transition to the new video (fading, folding, extending, etc) would be ideal.
- Content Browsing Scheme that allows users to select different parent videos to view.

Extra Features

The following are features that are unnecessary for functionality of the final submission, but do contribute to a full featured user experience.

- Audio response support
- Image response support
- Textual response support
- Ability for users to delete responses.
- Admin tools.
- User accounts and user validation(facebook integration, instagram integration, captcha).
- User profiles.
- Authenticate users with other social media accounts

Development Schedule

2016-10-27: Milestone 1 − Proof of Concept

For a Proof of Concept the following features will be implemented:

- A web-based video player capable of switching between video files quickly and resuming playback of original content.
  - Jordan Yelloz
- A preliminary functional user interface for displaying video Topics and a list of Responses
  - Francis Lagueu Fogang
  - Abhishek Mallepalli
  - Mikael Mendoza
- A rudimentary information storage and retrieval system capable of processing uploaded video files, placing them in structured storage units to allow videos to be retrieved by the video player.
  - Valera Maniuk

2016-11-10: Milestone 2

The next milestone will focus on redeveloping the proof of concept with proper software design practices. Furthermore, several features will be developed for the system.

- The design of the information storage and retrieval system will be finalized and the system will be implemented. This will be implemented as a REST web service capable of the following actions:

    - Clients will be able to fetch a list of top-level topics along with related metadata including the related web address to view the topic, the location of thumbnail images for the topic.
    - Clients will be able to fetch a list of responses to a specific topic along with similar metadata to the topic list.
- All content assets (videos, thumbnails) related to a content entry (Topic or Response) will be retrievable by clients through a static web repository.
- The existing user interface will be updated to maintain integration with formal services.

2016-11-22: Milestone 3

The third milestone should be a feature-complete milestone, mainly focusing on implementing the full client-side user interface.

- The Topic upload UI flow should be completely implemented.
- The Response upload UI flow should be completely implemented.
- The Content playback UI should be implemented.
    - Response counts should be visible on the current video's playhead UI component and updated reflecting the amount of responses near the current playback position.
    - The Response list contents should update during Content playback, reflecting the optimal Responses near the current playback position.
    - Responses selected by the user should be played back as quickly as possible

2016-12-01: Final Product

The final milestone should be a bug-fixing and enhancements iteration of the previous milestone.

Algorithms and Implementation

Response Refreshing & Playing (implemented by Mikael A. Mendoza)
This feature entails fetching the points during a Video Topic at which there are Responses, and refreshing the Response Panel with information about each Response at that point in the Video Topic. To do this, using Polymer's element object, a property was set called 'response_points', consisting of an Array of Numbers, each representing a point in the Video Topic where there exist Responses. Using Javascript's *setInterval* function, the Web Application is able to run a function every 5 seconds, and the Web App checks the current time in the Video Topic and see if there are any Response Points close enough to be relevant; if there are, the Response Panel is refreshed to show the Responses. Once there are responses on the Panel, the user can click on them and play them on the Video Player; to do this a Javascript *eventListener* was used, to fire a function when a Response was clicked on. The *eventListener* listens for user's commands, in this case a 'click' and calls the specified function to handle the Web Application's

response. The Web App can then switch the video being played to the Response selected, and afterwards returns to the point in the Video Topic where it left off.
Given that this type of Web Application is not widely available, this features and their implementation was devised by the developers, using standard practices in Web Development and Interactive Systems, such as Event-Handling and Listening.

Video.js Polymer Integration (Implemented by Jordan Yelloz)

The Video.js web multimedia framework, including the DASH file format playback support was integrated into a Polymer web component for this project. The goal was to provide a Polymer web component that contains a single Video.js instance that is re-used during the playback of multiple videos. This component takes advantage of the Polymer framework's data binding and property features so setting a property on the video player component allows the content to be switched out automatically. This functionality is achieved first by successfully wrapping the Video.js player object inside the life-cycle of a Polymer component. In the ready() method of the vlogme-player component, the Video.js player DOM element is accessed using a DOM selector and aliased into the component as a member variable and Polymer's scopeSubtree() method is called on the element which prevents Polymer's special CSS scoping mechanisms from controlling the non-Polymer Video.js element. Next, the attached() and detached() methods of the vlogme-player component were implemented in order to instantiate and destroy the Video.js element when the player object is added or removed from the page. The addition and removal logic was not necessary for this application but it would be noticed when re-using this component in a web application that creates and destroys many players.

This component also takes advantage of Polymer's data binding and URL routing features in order to allow external Polymer components to easily switch the content of the player by updating the video_uri and video_title properties of the vlogme-player component. These properties can also be bound to other properties (typically those of parent Polymer components) so the modification of external properties will reflect on those linked to the player component. This functionality was used by the top-level vlogme-app component to allow navigation within the single-page web application by allowing the entry of addresses such as /watch/<video-id> to actually display the video associated with <video-id> and programmatic property changes which can be used to facilitate swapping content when displaying the response to the current video.

Custom Playhead (Implemented by Abhishek Mallepalli

It was necessary to create a custom playhead as top level responses to a topic video are tied to a specific time during the video. It was necessary to create a flag at that point that indicates that there is a response there, and that users have the option to view the response and in doing so pause the video.

The playhead uses the default Video.js playhead with modifications that change the default time flag to instead display the number of responses at a point. The UX element uses Polymer bindings to retrieve the necessary data from the vlogme-app component and the vlogme-player component.

Custom Markers on the player(Implemented by Francis Lagueu Fogang)

I implemented the custom markers on the player using a videojs extended library called videojs-markers. It display customizable markers upon progress bars of videojs players. It could be used to show video breaks and show overlaid text on the video playback reaches the specific breakpoint. Some of the key features of the library are:

- Display markers on progress bar, with hover-over tooltips
- Display break overlays
- Flexible styling
- Support dynamically adding and removing markers

All this is done using JavaScript, which load a custom function which take multiple functions that allow you manipulate and customize the video object from Videojs. This function called *markers*. It take a *markerStyle* taking some CSS to style the marker. It also take a *markers* array containing multiple object to add breaks in the video by simply adding a new time in the list of breaks option. Another object it takes is the *markerTip* allowing to display time and text when hovered over the marker using the text and time function. It contains functions to trigger event on the markers such as *onMarkerClick*, and *onMarkerReached.* This library offer an API plugins that contains multiple functions to add and remove markers, move to the next and previous markers reset and destroy markers.