Developer’s Guide

By Meteoria

Table of Contents

1 INTRODUCTION

1.1 Description

Zoomable is a video streaming service that allows people to watch videos uploaded by the content provider, using a customized HTML5 canvas video player that supports zoom and pan functionality. This open-source service provides a user interface, Dashboard, to provide analytics tracking of the videos for the content provider.

1.2 Purpose

The purpose of this developer's guide is to provide the documentation of the design and development of Zoomable. The guide focuses on the base level of functionality of the system and is intended for developers who want to work on the project.

1.3 Glossary of Terms

Terms	**Definition**
Architecture Diagram	A graphical representation to show dependencies between software components.
Entity Relationship Diagram	A graphical representation to show relationship between objects.
Use Case	A list of actions between the actor and the system.
Model View Controller	A design pattern that separates a system into three components: model, view, and controller. Model handles the data logic, view displays the data, controller get requests from view and update model and view.
Client	Requests service from the server. Refers to Video Player and Dashboard.
Server	Provides service to client.
Content Provider, Content Manager	User who has account access to Zoomable.
Repository	A file archive used to store the code base for Zoomable.
System	Client and server working together to provide a service.
Viewer	User who views video using Video Player and has no

	access to Zoomable.
Dashboard	The interface for Content Manager to manage and upload videos.

1.4 Overview

The developer's guide is divided into 8 sections. Section 1 describes the purpose and overview of the developer's guide. Section 2 discusses the software requirements. Section 3 contains the system design and implementation. Section 4 includes the software development and installation steps. Section 5 covers the testing for backend and frontend development, continuous integration and code coverage. Section 6 discusses the technical issues in developing the Video Player. Section 7 shows the repository.

2 SOFTWARE REQUIREMENTS

The software requirements of this project is split into two main components: Functional requirements and non-functional requirements. For the functional requirements, sub-components have been further defined into 3 categories, namely: User, Server and Client. User refers to either the Content Manager and Viewers. Server refers to the video server used to store and process the videos uploaded by Content Manager. Client refers to the DASH-compliant zoomable Video Player.

2.1 Functional Requirements

User stories and user cases were used to capture functional requirements. The list of user stories can be found in Appendix A.1 and the full list of use cases can be found in Appendix A.3.

[User]

Content Manager

No.	Done ?	Requirements	Details
1	✓	Content Manager should be able to log in and out of dashboard.	
2	✓	Content Manager should be able to upload one or more high resolution videos (from 1280px * 720px to 4K) and see upload progress of the selected videos.	Supports up to 1080p instead of 4K
3	✓	Content Manager should be able to delete one or more videos in dashboard.	
4	✓	Content Manager should be able to view a list of uploaded videos in dashboard.	
5	✓	Content Manager should be able to sort the video list based on popularity (most viewed), privacy mode (public, unlisted) and date added (oldest, newest).	
6	✓	Content Manager should be able to search video by the video title.	
7	✓	Content Manager should be able to embed Video Player into another website.	
8	✓	Content Manager should be able to edit video information (title, description, tag, privacy) of all uploaded videos.	

9		Content Manager should be able to add subtitles for a video.	
10	✓	Content Manager should be able to see view-related statistics.	

Viewer

No.	Done ?	Requirements	Details
1	✓	Viewer should be able to play the video from the Video Player.	
2	✓	Viewer should be able to pause the video from the Video Player at any point of time within the duration of the video.	
3	✓	Viewer should be able to replay the video when the video has finished playing.	Some videos have an issue with ending
4	✓	Viewer should be able to see the time display (current time, time duration) of the video from the Video Player.	
5	✓	Viewer should be able to put the video on fullscreen mode.	
6	✓	Viewer should be able to zoom in and out of the video using the Video Player controls with and without full screen mode.	
7	✓	Viewer should be able to pan the video in any direction when in zoom mode.	
8	✓	Viewer should be able to see a minimap in zoom mode.	Minimap always present
9	✓	Viewer should be able to skip to a certain part of the video by dragging along the seek bar or clicking at a point on the seek bar.	
10	✓	Viewer should be able to adjust the volume of a video in the Video Player.	
11		Viewer should be able to share the video when the video has finished playing.	Not implemented
12		Viewer should be able to see a thumbnail of the video at a point on the seek bar.	Not implemented
13	✓	Viewer should be able to take and download a snapshot of a point in the video.	
14		Viewer should be able to show or hide subtitles for a video.	Not implemented
15	✓	Viewer should be able to close the video at any point in time.	

[Server]

Video Server

No.	Done?	Requirements	Details
1		Video Server should ensure the uploaded videos are not corrupt.	
2	✓	Video Server should be able to re-encode uploaded videos into different resolutions, and store these different versions in addition to the original video.	
3	✓	Video Server should be able to split uploaded video into tiles, and store these tiles in addition to the original video.	
4	✓	Video Server should be able to store a high resolution video (from 1280px * 720px to 4K) with a title.	Supports up to 1080p instead of 4K
5	✓	Video Server should be able to gather video statistics from the database and provide statistics visualization.	
6		Video Server should be able to detect disconnection by the Video Player and abort streaming to that Video Player.	

[Client]

Video Player

No.	Done ?	Requirements	Details
1	**?**	Video Player should deliver any stream that is compatible with the DASH-264 specification.	
2	✓	Video Player should detect Viewer's actions and change the resolution or viewport of the video accordingly	
3	✓	Video Player should provide a downloadable image file for any frame in the video, when prompted by the viewer requesting a snapshot of that moment in the video.	
4	✓	Video Player should detect network quality and change the video quality accordingly.	

5	✓	Video Player should, in zoom mode, display a minimap which indicates the current viewport.	Minimap displays even when not in zoom
6		Video Player should be able to send a 'Disconnect' signal to the Video Server to stop streaming.	
7		Video Player should synchronize all streams / tiles to ensure smooth playback of the video.	

2.2 Non-Functional Requirements

We propose a list of misuse cases and abuser stories to handle security risks. Abuser stories can be found in Appendix A.2 and the misuse cases can be found in Appendix A.4.

[System]

No.	Done?	Requirements	Details
1		Scalability	Video Server should be able to handle at least 10000 concurrent connections.
2	✓	Reliability	Video Server should be able to have 99% of continuous uptime.
3	✓	Documentation	There should be extensive developer documentation provided.
	✓		There should be well designed and documented APIs provided.
4	✓	Security	Video Server should be able to handle brute force attack aiming to get unauthorised access by using reCAPTCHA verification method.
	✓		Video Server should be able to prevent code injection by validating user inputs before being writing into the database.

[User]

No.	Done ?	Requirements	Details
1	✓	Usability	User should be able to perform a use case with less than 6 steps.
2	✓	Learnability	User should be able to perform any use case within 10 minutes after a 30 minutes' introduction to the system.
3		Performance	Selected video should load (starts playing) within 5 seconds.

4		Testability	System should include unit tests that covers at least 90% of the use case.
5		Robustness	System should be able to handle unexpected actions and abnormal termination from the user.
6		Error Handling	System should catch at least 90% of runtime exceptions.
	✓		System should check for errors from user input and show relevant error messages back to user.

3 DESIGN AND IMPLEMENTATION

3.1 System Architecture Overview

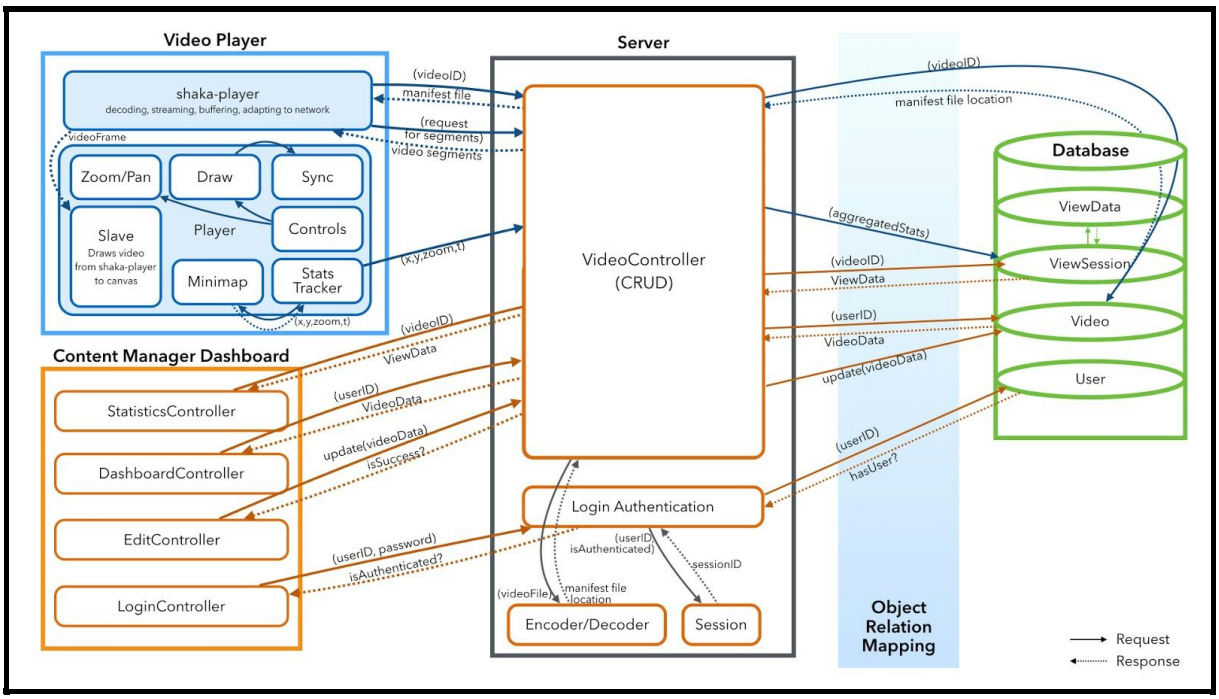


Figure 3.1.a. Architecture Diagram for Zoomable

Our system can be divided into 4 main components, namely, the Video Player, the Content Manager Dashboard, the Server and the Database. Due to the complexity of our system, we have broken down the components accordingly into smaller subcomponents, as shown in Figure 3.1.a.

Our Video Player is mainly built on top of Shaka Player and canvas. We added more features such as Minimap, Zoom/Pan as well as Stats Tracker. The Stats Tracker is responsible of sending

user action data such as the zoom factor and the coordinates of the video view. The Video Player only interacts with the Streaming and Stats Manager subcomponents in the Server.

The client, the Content Manager Dashboard, is an administrative panel for content providers to manage their uploaded videos. This includes features such as uploading and editing video information and viewing of video statistics. This component only interacts with Video Controller and Login Authentication subcomponents in the Server.

Our Server interacts with all the main components of our system. One important feature of our server is that it automatically encodes the uploaded videos into different resolutions and processes the video to be compatible with the Player, via operations such as tiling and segmentation. This makes the uploaded videos compliant to the DASH standard and allows the Video Player to utilize DASH protocol in playback. All the data is stored inside the Database.

Our Database can be subdivided into 4 subcomponents, mainly ViewSession, ViewData, Video, and User. The Server interacts with the Database via the Object Relation Mapping. Further explanation on the Database model is discussed in Section 3.1.4.

3.1.1 Content Manager Dashboard

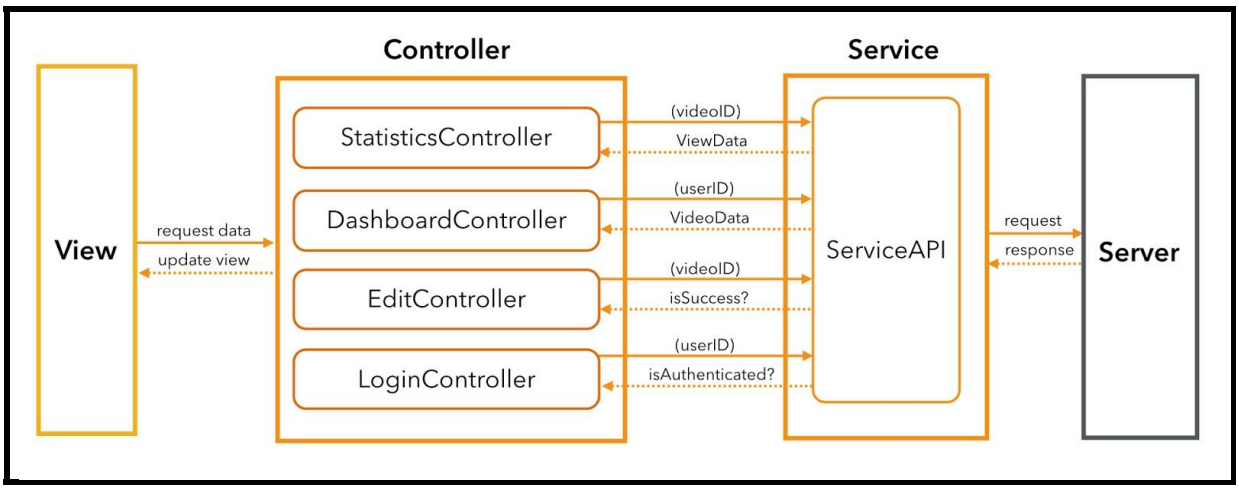


Figure 3.1.b. Content Manager Dashboard Component within Architectural Diagram

The Content Manager Dashboard component follows a Model View Controller (MVC) principle. The Controller handle requests from the View and communicate with the Server using a Service. The Server then updates the Model accordingly. Controller consists of DashboardController, EditController, LoginController and StatisticsController.

DashboardController

DashboardController supports filtering, sorting, uploading and deleting of videos. It uses a Service, ServiceAPI, to get and update video information from the Server and displays them in the View.

`getVideoList()`	Call `ServiceAPI.get()` to retrieve a list of video belonging to the user.
`updateSortState()`	Sort the list of videos by latest date or most number of views.
`updateFilterState()`	Display videos accordingly to their mode of privacy.
`upload()`	Check if video file belongs to supported types: `MP4` or `MOV`. If video files are not supported, it will prompt an 'unsupported file format' error message. If video files are supported, it will call `ServiceAPI.upload(videoFile)` and display a progress bar to indicate upload progress in the View.
`getProcessStatus(videoId, index)`	Create an instance, $interval, which call `ServiceAPI.getUploadProgress()` to check for the processing status of video by `videoId.` Update video list in view to display video thumbnail once processing is completed. If there is an error in the processing the video, it will prompt an 'error processing video' error message in the View.
`showConfirmDeleteByButton()`	Delete a single video by calling `ServiceAPI.delete(videoID)`.
`showConfirmDeleteByCheckbox()`	Delete videos selected by checkboxes by calling `ServiceAPI.deleteAPI(videoIDs)`.

EditController

EditController update the video fields if user makes new changes in the View.

`showConfirm()`	Prompt a dialog if user has unsaved changes.
`saveChanges()`	Validates fields updated by user in the View. Call `ServiceAPI.update(videoID, fields)` to update the video fields.

LoginController

LoginController validates user credentials and directs user to the video list page. It also controls the behaviour of the navigation bar and sidebar.

Login Authentication	
`submitForm()`	Check if user is creating a new account or logging in. Create new user account by calling `servicesAPI.createAccont(accountData)` Authenticate user by calling `servicesAPI.login(loginData)`.
`hasEmptyFields()`	Check if required fields are filled.
`toggleLeftMenu()`	Toggle sidebar.
`searchVideoList()`	Catch strings in search input in navbar. Emit a broadcast to DashboardController to update filter type.

StatisticsController

StatisticsController generates the heatmap visualisation and view analytics for the videos.

Heatmap Visualisation	
`getVideoViewData()`	Get a list of view data by calling `servicesAPI.getHeatMapStats(videoID)`.
`generateHeatmapVideo()`	Add a canvas context into an array of canvas contexts for each second.
`coordinatesPerSecond()`	Return a canvas context with aggregated coordinates (x,y) of view area.
`generateCanvas()`	Create a heatmap canvas with coordinates using third party library heatmap.js.
`compileHeatmapVideo()`	Compile heatmap video using third party library Whammy.js which converts canvas into video.
View Analytics	
`processViewSessions()`	Order view sessions and count by date.
`setStartAndMinDate()`	Set start and minimum date for statistics.
`updateCriteria(), updateDate()`	Set date criteria by day, week, week.

3.1.2 Video Player

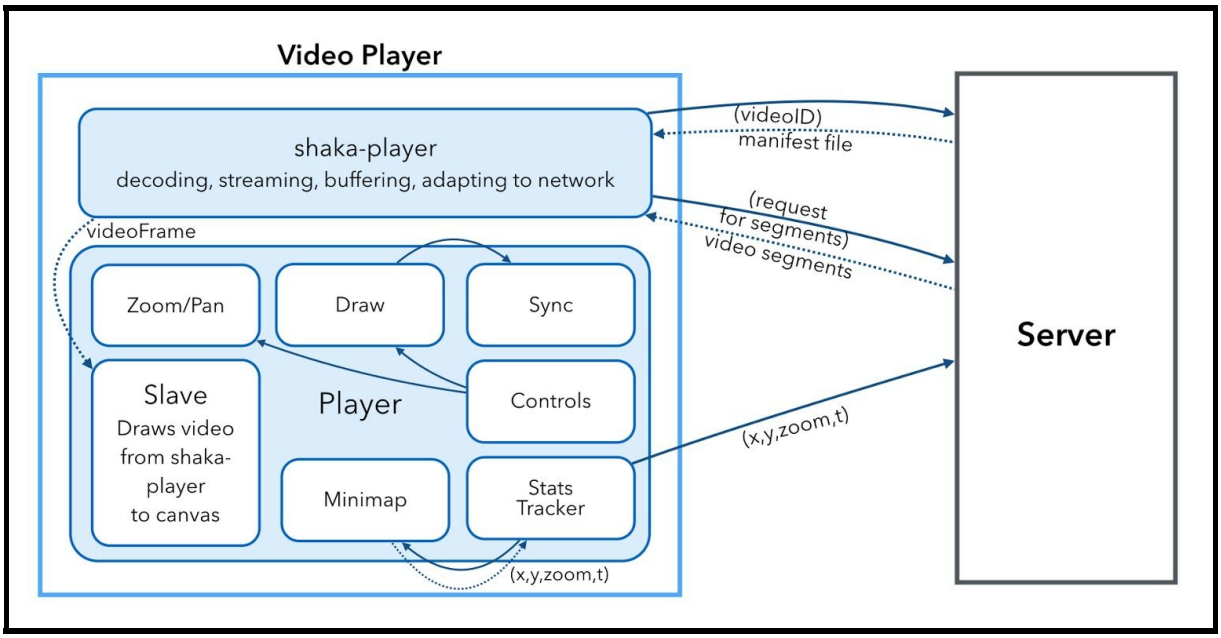


Figure 3.1.c. Video Player Component within Architectural Diagram

The Video Player component of the architecture deals with the playback of videos, along with additional functionality of zooming and panning. We use Shaka Player for all basic player functionality, such as buffering and processing the various MPDs, as well as handling network adaptation. The Player sub-component consists of other sub-components, as indicated in Figure 2, such as Zoom/Pan, Draw, Sync, Controls, Stats Tracker, Minimap and Slave.

Zooming and Panning

The Player maintains a Singleton `Transforms()` instance that keeps track of the transformation matrix of the canvas context and provides the base transform functions applied to the canvas.

`Transforms()`	– Initialises a svg object for xform to modify – Adds a method `.save()` to the Canvas context that saves the context transform. – Adds a method `.transformedPoint()` to the Canvas context that tracks the last modification
`.savedTransforms`	An array containing the stack of transform matrices
`.xform`	Contains the transformation matrix [[a c e][b d f][0 0 1]] through the properties (*a, b, c, d, e, f*)
`.restore()`	Pops a transform matrix from the stack, reverting to the previous transformation.

`.scale(sx,sy)`	Scales the current context transformation by *sx* horizontally and *sy* vertically.
`.rotate(r)`	Rotates the current context transformation by *r* radians (not used since our Player only allows translation and scaling)
`.translate(dx,dy)`	Translates the current context transformation by *dx, dy*
`.transform(a,b,c,d,e,f)`	Multiplies the current `.xform` (transformation matrix) by [[a c e][b d f][0 0 1]] to ***apply*** the new transform to the ***current*** one.
`.setTransform(a,b,c,d,e,f)`	Sets the `.xform` (transformation matrix) to the [[a c e][b d f][0 0 1]] The new coordinates would be x = ax + cy + e, y = bx + dy + f, ***replacing*** the old transform.
`.refit()`	Checks if the viewport borders intersect with the canvas borders. If it intersects, then scale/translate back the canvas accordingly to fit the viewport. This prevents the user from being able to zoom in too far or pan past the video picture borders.
`.draw()`	*** This isn't actually in the code yet but maybe soon??? Lol LOLOL :D
`.redraw()`	Redraws the canvas whenever a zoom/pan action is done to display the transformed view, updates the viewport indication rectangle in the minimap. Send stats is also being done here, we need to change that.
`.outerTranslate()`	A wrapper Translate function that will also call `.refit()` upon translating.

Sync

The Player also maintains a singleton instance of `Sync()`, that keeps track of the time changes, pause states and end states, for each stream's video. These changes will be synchronized to ensure that all streams follow a common time, pause, or end state. Additionally, it is in charge of ensuring that all Slaves are displaying the same frame at any point in time, which is currently not implemented.

`Sync()`	Consists of 3 main functions to maintain a synchronized player time, pause state and end state.
`.currentTime()`	Iterates through an array that consists of each stream's `currentTime` property. This function will search for the earliest time and set the overall player's video time to the earliest one. If the difference between the video duration and the player's video time is within a

	certain threshold, a forced end state will be triggered.
`.pauseState()`	Iterates through an array that consists of each stream's paused property. This function will check if all videos are concurrently paused, before updating the overall player's pause state, to true or false. Upon updating the overall pause state, the UI control's button will be updated accordingly.
`.endState()`	This function checks if all videos have ended, or if the video has entered a threshold that requires invoking a forced end state, for the video to actually end. If the videos have ended, or if the video requires to be ended, then the overall end state of the player is set to true, and the UI controls will be updated to the 'Replay' state accordingly.

Controls

The Player also maintains another singleton function for the player's UI controls, updating its various buttons on the UI accordingly.

`Controls()`	Methods are the various functions invoked upon UI interaction. Properties contain the HTML elements of the player's UI: `.playPauseBtn`, `.currentTimeTxt`, `.totalTimeTxt`, `.seekCtrl`, `.volumeBtn`, `.volumeCtrl`, `.zoomOutBtn`, `.zoomCtrl`, `.zoomInBtn`, `.snapshotBtn`, `.fullscreenBtn`
`.uiControls`	The parent `div` element of the other button elements
`.playPauseVideo()`	This function is invoked when the player is paused, has ended, or is being played. Triggers the play / pause functions of the videos and audio, and updates the UI elements accordingly.
`.playVideo()`	Calls the `play()` function on each Slave's video.
`.pauseVideo()`	Calls the `pause()` function on each Slave's video.
`.restartVideo()`	Sets the `currentTime` for all the videos back to zero, and triggers the play function for all the videos.
`.changeToPauseState()`	Changes the `.playPauseBtn` to the 'Play' icon upon invoking the pause state.
`.changeToPlayState()`	Changes `.playPauseBtn` to the 'Pause' icon upon invoking the play state.
`.changeToReplayState()`	Changes the `.playPauseBtn` to the 'Replay' icon upon invoking the end state.

`.getVideoLength()`	Converts the video's duration to a format for displaying on `.totalTimeTxt`.
`.updateCurrentTimeTxt()`	Converts the player's time value to a format for displaying and updating on `.currentTimeTxt`
`.updateZoomUI()`	Reads in the zoom factor of the player and converts it to a proportionate value to update `.zoomCtrl`. Invokes `.updateSliderUI()` to update the slider UI for the zoom slider.
`.updateSliderUI()`	Updates the Slider UI to display the correct position/time.
`.takeSnapshot()`	Draws an image of the video player's canvas, to a separate canvas, and downloads that image to the user's computer.
`.toggleFullScreen()`	Toggles the fullscreen mode, based on the user's browser. Utilises webkit's functions to implement fullscreen mode.

Minimap

The minimap is a singleton class that handles the drawing of the minimap, including its borders and the zoomed portion of the user's view.

`Minimap()`	
`.player`	The Player object that it belongs to.
`.canvas`	The canvas on which the Minimap draws the full view of the video.
`.rect_canv`	The canvas on which the Minimap draws the rectangle that indicates the current viewport size and position.
`.video`	The HTML5 video element playing a small version of the unmodified video.
`.outline`	`.draw()`: draws viewport rectangle to `.rect_canv`
`.transforms`	`.draw()`: copies the full view of the video to `.canvas`

Stats

The Player utilizes a singleton class that contains functions that create the statistic information into objects, and functions that will send this object to the relevant address.

`.createStats(`	Creates an object with properties that are related to the statistics for

)	the heatmap and returns it.
`.sendStats()`	Takes in the object returned by `.createStats()` and creates a XMLHTTPRequest object to create a HTTP POST request and send the object as a JSON, to the relevant address.
`.statTrack()`	A wrapper function that invokes `.createStats()` and `.sendStats()`.

3.1.3 Server

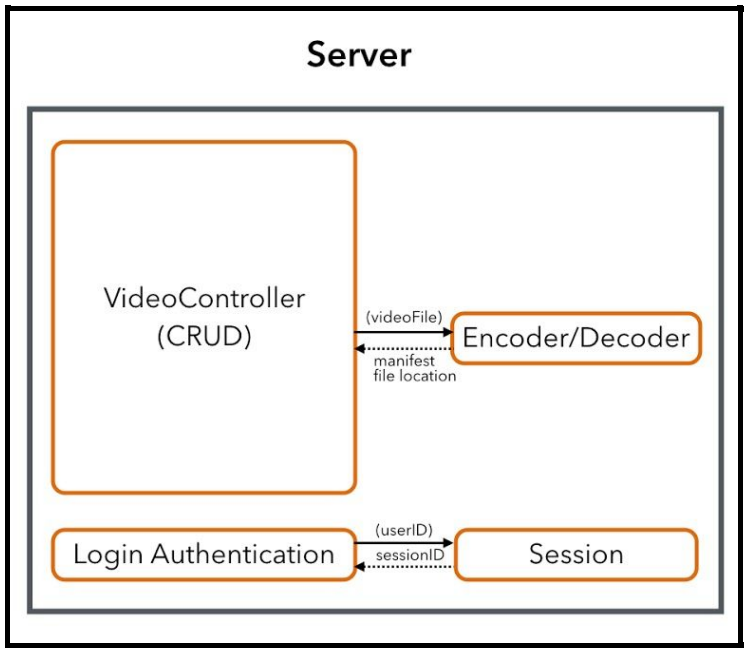


Figure 3.1.d. Server Component within Architectural Diagram

VideoController

VideoController handles the data logic and communication between the client and database. It handles the video view sessions and zoom data that is sent from the video player whenever a video is being viewed. The video player will make an API call sending a unique session ID whenever a new view is made, and this session ID will be sent together with the subsequent calls to collect zoom data from the player. This ensures that the zoom data for a video will always be associated to a specific viewing session for easy retrieval thereafter.

Encoder / Decoder

The encoder / decoder component handles the video processing, whereby uploaded videos will be decoded from their present format (currently we only support input MPEG-4 and MOV files) and re-encoded into several qualities (360p, 480p, 720p and 1080p), before cropping videos of each quality type, into 12 tiles. After which, different resolutions of the same tile will be

compiled into an MPD to allow for resolution switching during network adaption by the player. The audio is generated into a separate audio file, from the videos. The encoding and decoding is done through the FFMPEG library, while the MPD compilation is done via the MP4Box library.

FFMPEG Flag	Meaning
`-c:v libx264`	Encoding the output video to H.264
`-r 30`	Adjusts the video's bitrate to 30 frames per second
`-vf scale=wxh`	Scales the video to a new resolution, where w and h are the width and height of the video's new resolution, respectively
`-x264opts bframes=0:keyint_min=250`	Overwrites the default x264 encoding settings, to have 0 B-Frames and sets the minimum number of frames between 2 I-Frames.
`-movflags +faststart`	Moves the metadata of all the packets to the start to allow for faster playback.
`-preset slow`	Selects a 'slow' encoding speed in order to provide better compression and better quality for the video.
`-profile:v baseline -level 3.0`	Limits the profile to the baseline H.264 profile that is compatible with all target devices (Refer to FFMPEG documentation for a full list of devices)
`-an`	Disables the audio for the video output

Login Authentication & Session

The login authentication component makes sure users are authenticated without saving the user password in the database. The component uses BCrypt.js which helps to hash the password and the database save the hashed password in the database. When the user wants to login, the input password is hashed with BCrypt.js and compared with the stored hashed password. This method will ensure users password are stored securely. Session is maintained by Node Session. When the user is authenticated, the session cookies will be stored in the session memory store of the backend system. The user is able to carry out any APIs which requires authentication once his/her session is verified and stored.

3.1.4 Database

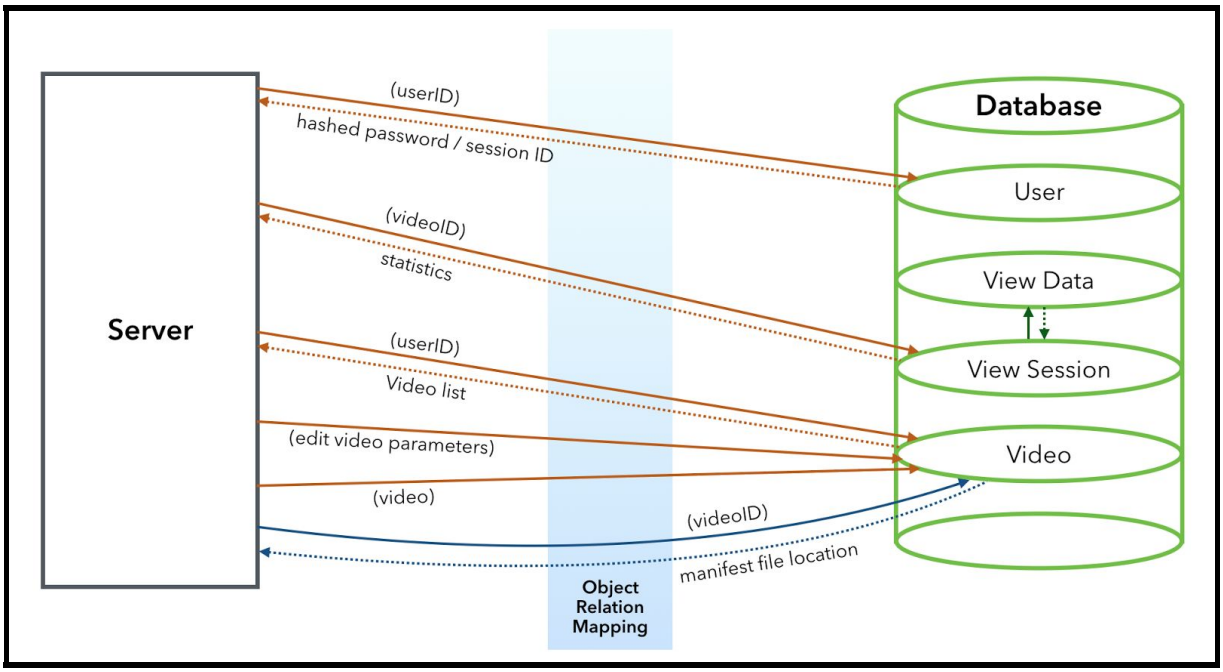


Figure 3.1.e.Database Component within Architectural Diagram

The Database component of the architecture deals with all the data required for Zoomable. It involves an object relation mapping (ORM) called Waterline. Waterline provides a set of model methods which we use to manipulate and query our database.

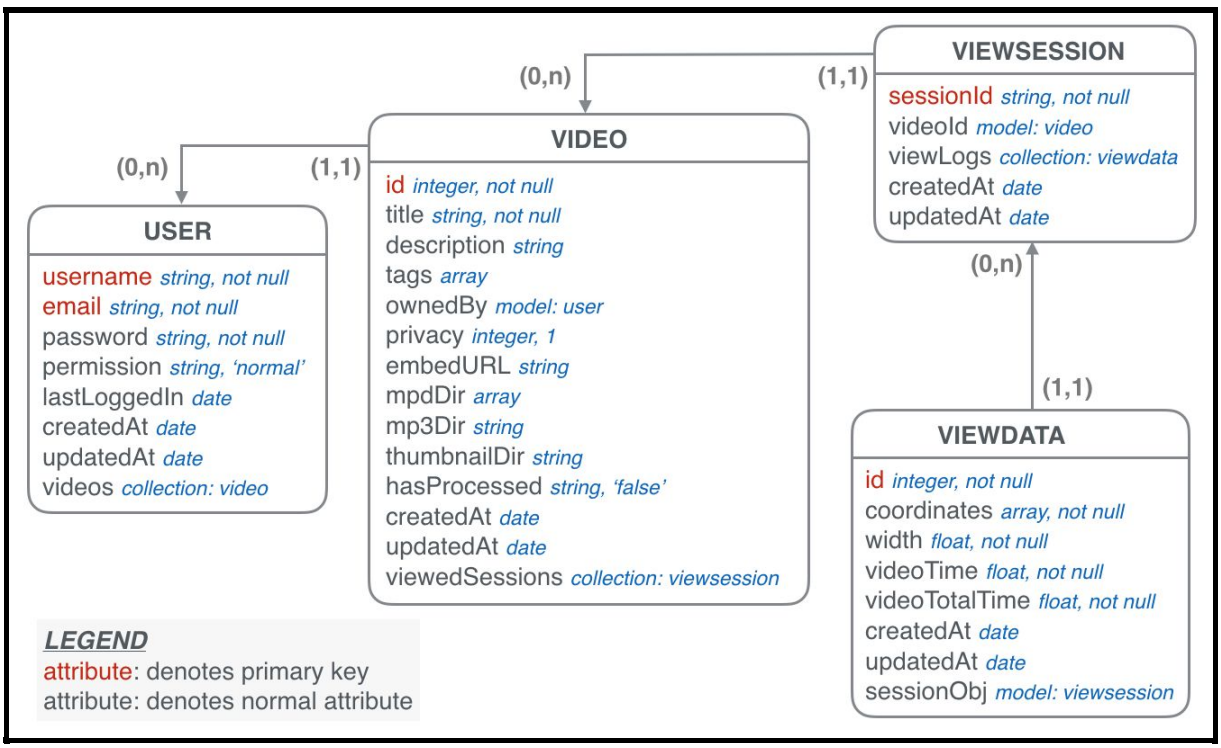


Figure 3.1.f. Entity Relationship (ER) Diagram

With reference to the ER diagram in Figure 3.1.f, we have 4 main models for our database, each containing different types of records. These models are defined in `api/models` directory.

The User model contains the Content Manager records. Content managers are identified uniquely by their email and username. The User model has multiple one-to-many associations on the Video model, where each Content Manager can have none or multiple videos linked via the `videos` attribute.

The Video model contains the video records uploaded by the Content Manager. Each video is identified uniquely by its `id`, and is always associated to a Content Manager via the `ownedby` attribute. The Video model has multiple one-to-many associations on the View Session model, where each Video can have none or multiple view sessions linked via the `viewedSessions` attribute.

The ViewSession model contains the session records that are created when a video is viewed. Each view session is identified uniquely by its sessionId, and is always associated to a video via the `videoId` attribute. The ViewSession model has multiple one-to-many associations on the View Data model, where each ViewSession can have none or multiple view data linked via the `viewLogs` attribute.

The ViewData model contains the view data records for each of the view sessions. Each view data is identified uniquely by its `id`, and is always associated to a view session via the `sessionObj` attribute. The view data records store information about zoom level at the specified video playback time. These records are subsequently used to generate the heatmap videos for zoom frequency analysis.

3.2 API Documentation

3 main APIs are provided within this project: User API, Video API and ViewSession API. These APIs are defined in `api/controllers` directory.

User API

1. Login to account

URL	`/api/user/login`
HTTP Method	POST
Required Inputs	username: registered user's username password: registered user's password associated with username

2. Create a new account

URL	`/api/user/signup`
HTTP Method	POST
Required Inputs	email: unique email address not currently registered in system username: unique username not currently registered in system password: password associated to new username ** username and password must contain at least 6 characters*

3. Logout of account

URL	`/api/user/logout`
HTTP Method	GET
Required Inputs	-

4. Get user's account creation date

URL	`/api/user/getAccountDate`
HTTP Method	GET
Required Inputs	-

Video API

1. Add a video entry

URL	`/api/video`
HTTP Method	POST
Required Inputs	title: title of the video

2. Retrieve a video

URL	`/api/video/:id`
HTTP Method	GET
Required Inputs	id: id of the video

3. Retrieve a list of video

URL	`/api/video`
HTTP Method	GET
Required Inputs	-

4. Delete a video

URL	`/api/video/:id`
HTTP Method	DELETE
Required Inputs	id: id of the video object

5. Delete multiple videos

URL	`/api/video`
HTTP Method	DELETE
Required Inputs	id: array containing list of video id to delete

6. Edit information of a video

URL	`/api/video/:id`
HTTP Method	PUT
Required Inputs	id: id of the video object

7. Upload a video

URL	`/api/video/upload`
HTTP Method	POST
Required Inputs	id: id of the video object video: video file to upload

8. Get processing status of a video

URL	`/api/video/isComplete`
HTTP Method	POST
Required Inputs	id: id of the video object

9. Get view session of a video to calculate video views

URL	`/api/video/getStat/:id`
HTTP Method	GET
Required Inputs	id: id of the video object

10. Get view session statistics for all videos to calculate total video views

URL	`/api/video/getStats`
HTTP Method	GET
Required Inputs	-

ViewSession API

1. Create a video view session

URL	`/api/viewsession`
HTTP Method	POST
Required Inputs	sessionId: unique id for session videoId: id of the video the view session belongs to coordinates: array containing x and y position of zoom width: width of video videoTime: current time of video when request is made videoTotalTime: total time of entire video

2. Get list of view data for a video

URL	`/api/viewsession/getVideoId/:id`
HTTP Method	GET
Required Inputs	id: id of the video object

3.3 User Interface

The application provides the Content Managers with a user interface to upload and manage videos, along with a video player that can be used to zoom and pan on a video.

Login Page

Content managers will have to create a new account and login to their dashboard in order to upload or view their videos.

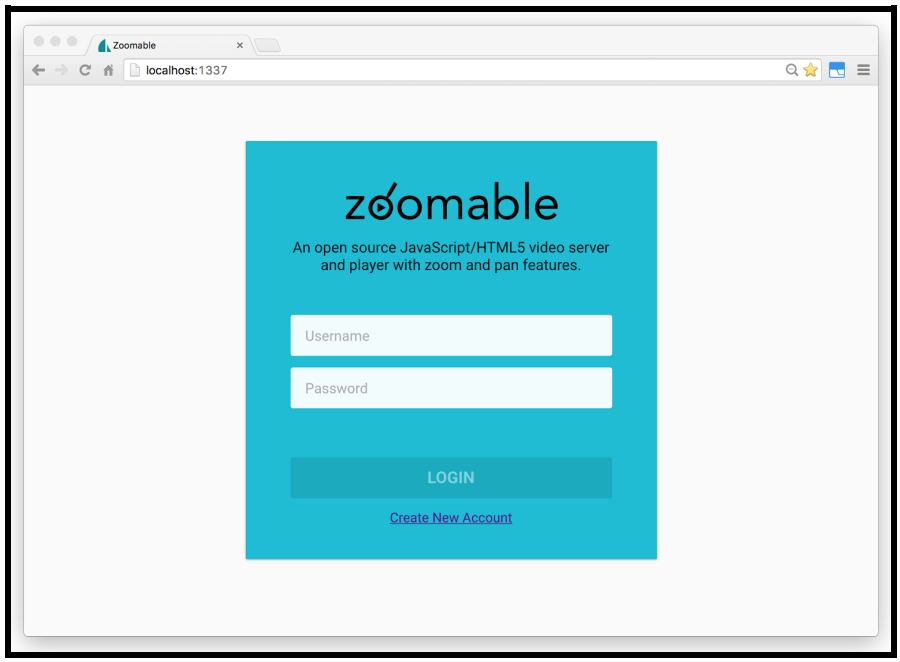


Figure 3.3.a. Login Page

Video List Page

The video list page is only accessible if the Content Manager has logged in. The list of videos uploaded by the Content Manager will be shown here, along with the option to upload, edit or delete a video.

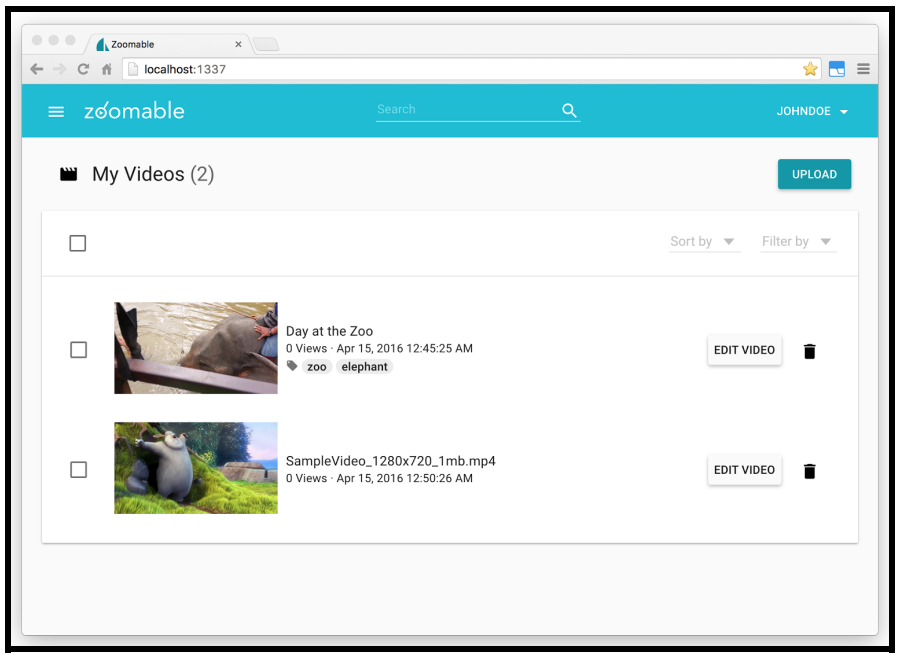


Figure 3.3.b. Video List Page

Upload Video

A Content Manager can upload videos by clicking on the UPLOAD button on the top right corner of the video list page. It will prompt a pop out dialog that supports drag and drop of multiple video files. Once he has selected a supported video file, the dialog will display the video title, video size and a progress bar indicating the upload status. A DONE button will be displayed at the bottom of the dialog to signal the completion of upload.

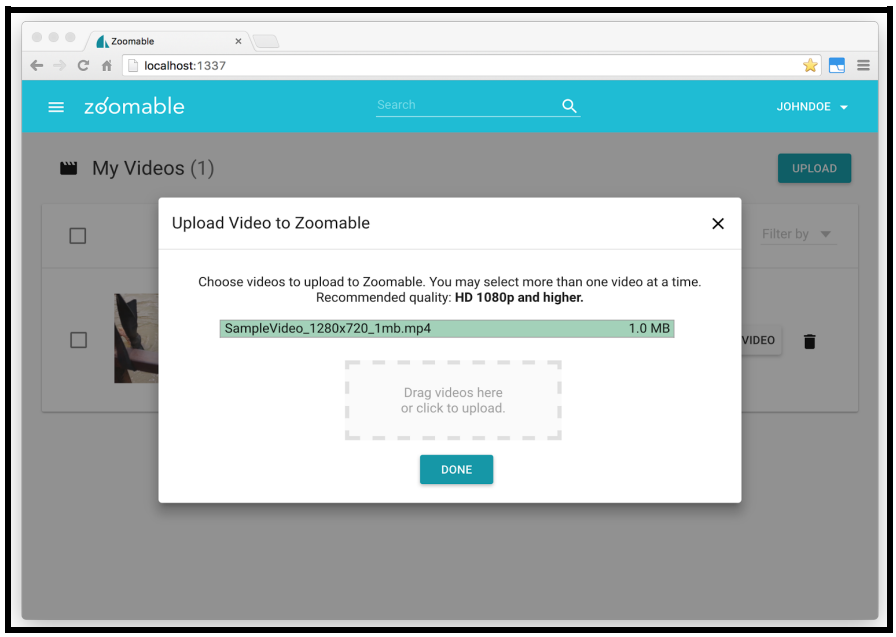


Figure 3.3.c. Upload Video Dialog

Edit Video

In INFO & SETTINGS tab, the Content Manager can update video title, description, mode of privacy and tags to categorise the video. The video player and embed link are shown after the video is processed.

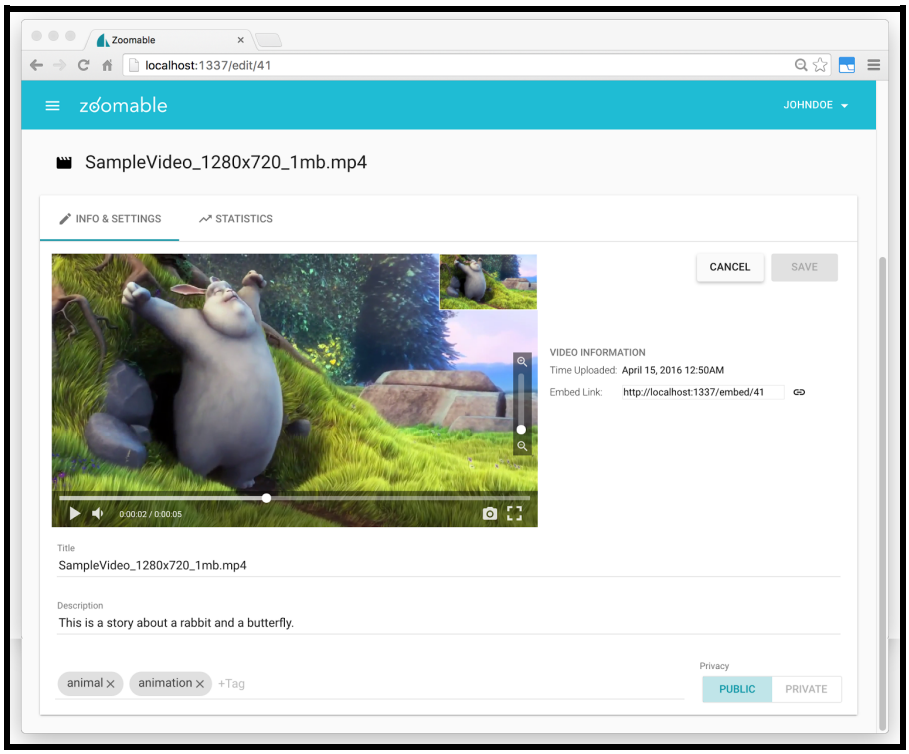


Figure 3.3.d. Info & Settings Tab

Video Statistics

In the STATISTICS tab, the Content Manager can view a heatmap video that indicates frequently zoomed area by viewers and a graphical representation of the number of views over a day, week or month.

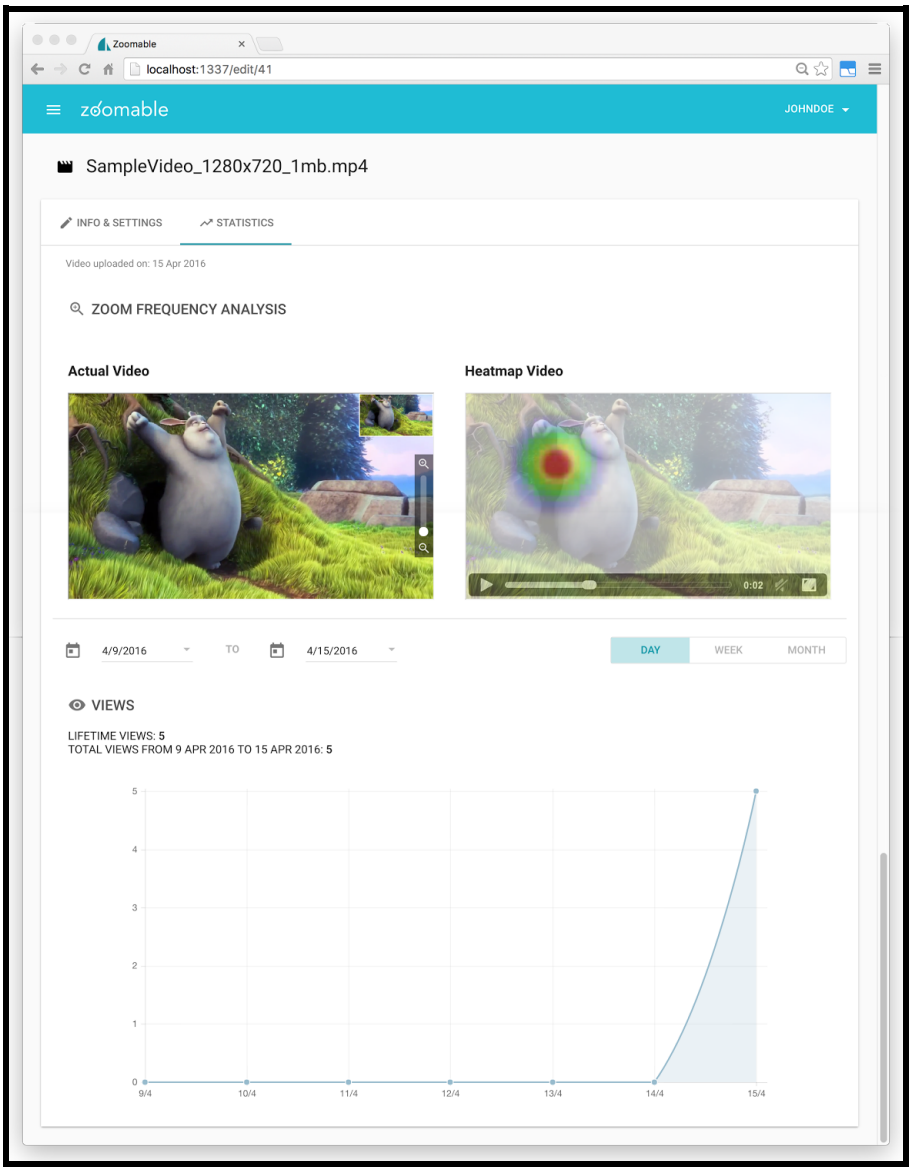


Figure 3.3.e. Individual Statistics Tab

Video List Page Sidebar

Zoomable provides a statistical overview of all the videos which can be accessed from the sidebar. Sidebar hides features that are not frequently accessed to prevent distraction. For future extension, shortcuts can be included in the sidebar to give the Content Manager freedom and control over the application.

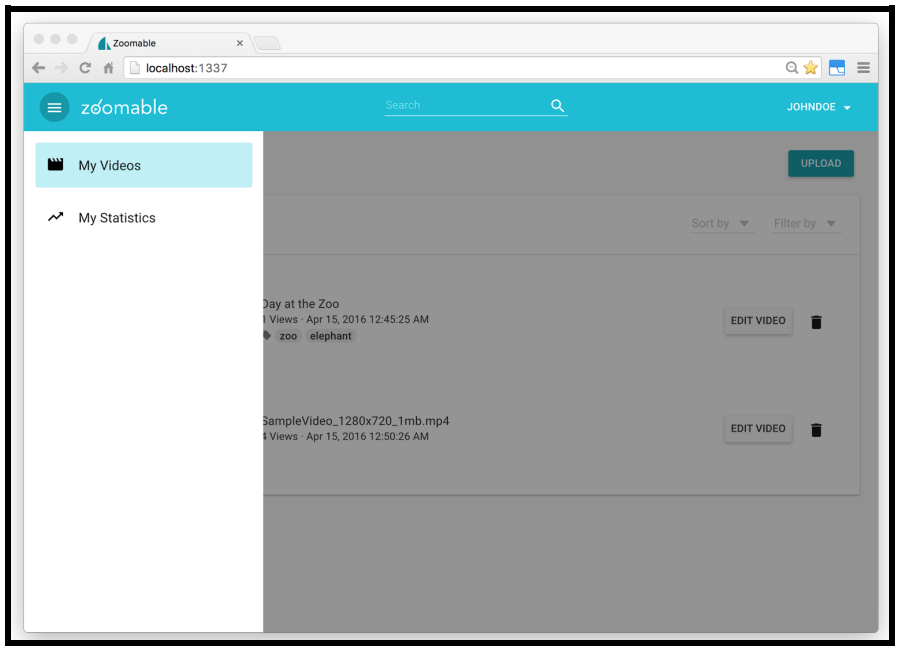


Figure 3.3.f. Sidebar toggled from Navigation Bar

Overall Statistics Page

The overall statistics page shows the number of uploaded videos, the date the Content Manager created the account, username and the number of views of all the uploaded videos in a graph. User can choose to view the statistics by day, week or month.

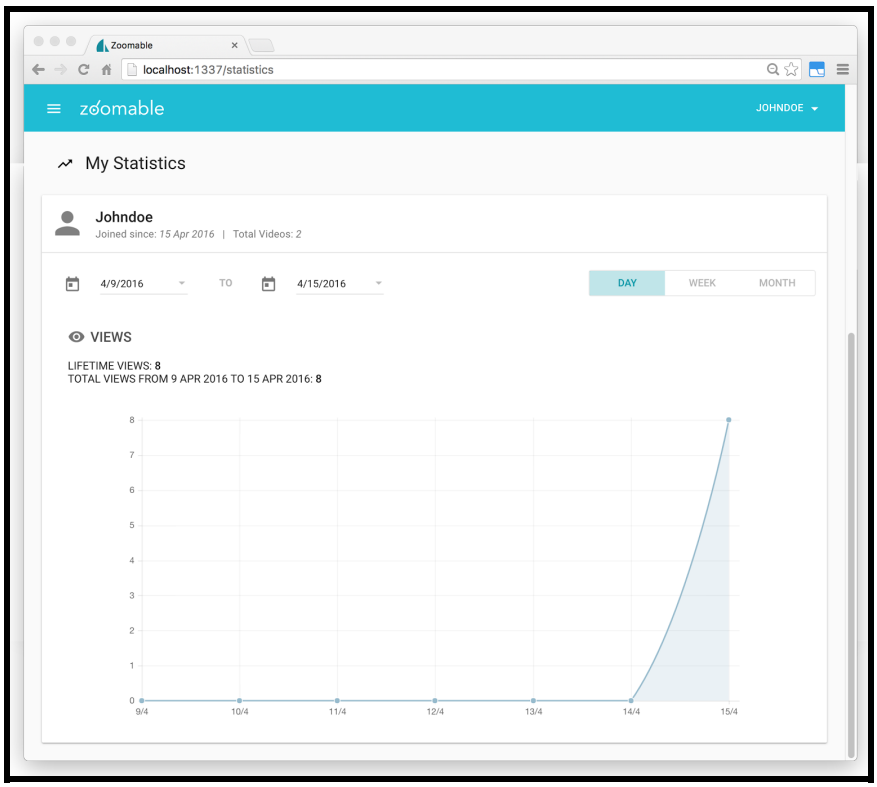


Figure 3.3.g. Overall Statistics Page (Video Views only)

Video Player

The video player is created using HTML5 canvas element, and the player controls are being placed on top of the video player. It features the basic player controls such as play, pause, replay, fullscreen and volume controls. The new controls that we have added is the ability to take a screenshot of the current frame, as well as buttons to control zoom level of the video. A zoom minimap is included at the top right hand corner of the player.

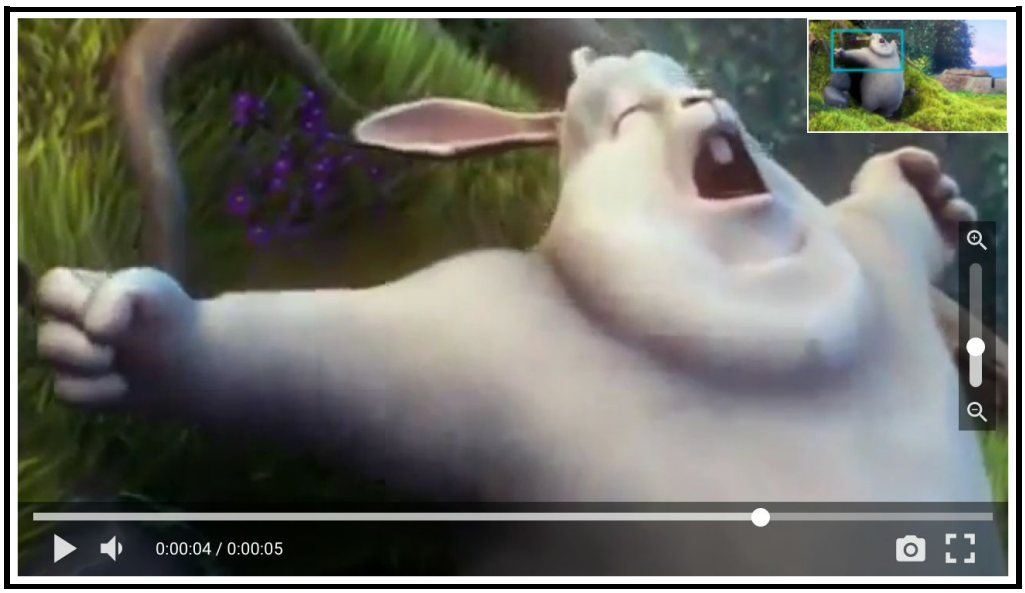


Figure 3.3.h. Video Player

4 SOFTWARE DEVELOPMENT ENVIRONMENT & INSTALLATION

4.1 Development Tools

The following sections describe the frameworks and libraries used by Zoomable.

4.1.1 Backend Development

Sails.js	Sails.js is used as our main backend framework.
PostgreSQL	PostgreSQL is used as our database.

4.1.2 Frontend Development

AngularJS	AngularJS is used as our main frontend framework to build the interface of Zoomable.
Angular Material	Angular Material is a UI Component framework used to design the user interface of Zoomable.
Angular Chart	Angular Chart is a library that provides a set of angular directives based on Chart.js, to create simple and responsive charts for our video views graph.
Heatmap.js	Heatmap.js is used for generating the heatmap frames for zoom statistics visualisation.
Whammy.js	Whammy.js is a WebM Encoder to convert the heatmap frames drawn on a canvas into a video.
Moment.js	Moment.js is a date library used to manipulate and format dates for video view statistics.

4.1.3 Video Player

Shaka Player	Shaka Player is an open-source JavaScript library which implements a DASH (Dynamic Adaptive Streaming over HTTP) client. Our HTML5 canvas video player is built on the Shaka Player.

4.1.4 Revision Control

Git	Git is a distributed version control system. We use GitHub, a third party service, to easily manage our code and collaborate among the team.

4.2 Setting up the Development Environment

4.2.1 Using Docker

In order to speed up the development process, Docker is used to provide a unified development experience. A Dockerfile is included in the repository to aid new developers to build a standard Docker image for development. After installing Docker, you can build the image:

```
$ docker build -t <your-username>/<repository-name> .
```

After successfully building the image, you can access the built image:

```
$ docker run -it -p 1337:1337 --entrypoint='bash' <your-username>/<repository
  -name>
```

No extra configurations are needed and you are free to start developing for Zoomable.

4.2.2 Without Using Docker

To set up the development environment, you need to install the following dependencies:

- NodeJS
- SailsJS
- FFMPEG - For video conversion, thumnbnail generation
- MP4Box - For MPD creation for the videos

Then, install the necessary packages using npm and bower install:

```
$ npm install -g bower grunt-cli
$ npm install
$ bower install
```

One last additional configuration is to make the video-processing script executable.

```
$ cd scripts
$ chmod +x video-processing.sh
```

One important thing to note is that the video-processing script is using the default `/bin/bash`. Please change to appropriate path if your system is configured differently.

4.3 Development Workflow

We have 2 main branches on Github, master and develop. We use the master branch for rolling out new versions of Zoomable. For our daily work and development, we use the develop branch. The standard workflow is as follows:

1. The developer creates a new branch from develop, and does all work on that branch.

2. After the intended feature is complete, the developer commit and push his changes to the new branch.
3. The developer submits a pull request to merge the new branch into the develop branch.
4. Other team members then conduct code review to ensure code quality and correctness.
5. If the code review goes well, the pull request is then merged and marked as closed.
6. The new branch can be subsequently deleted.

4.4 Setting up Production Environment

Before setting Zoomable to production mode, there are some settings needed to be done. The config/local.js can be used to configure the local environment variables, in particular storing database settings, change the port used when lifting the Zoomable app etc. Since `config/local.js` is being listed in .gitignore file, a `config/local_sample.js` is provided for you to modify.

Firstly, you need to set up your choice of database system. The connection adapter provided in `config/local_sample.js` and `config/env/production.js` is PostgreSQL but you are free to change it to any supported Sails adapter. You need to create a new user and database for Zoomable in PostgreSQL.

Secondly, you need to get a set of SSL certificate and key to serve HTTP responses over https://. Place the set of SSL certificate and key in the `config/env/ssl` folder. You then need to change the filename of the key and certificate in the config/env/production.js.

Thirdly, you need to change the appUrl to your domain in `config/env/production.js`. With all these setting set, you are ready to launch the application in production model.

4.5 Coding Convention

We follow the Google JavaScript coding convention. Some of the important rules are:

1. Naming Convention
    - Use functionNamesLikeThis, variableNamesLikeThis, ClassNamesLikeThis, EnumNamesLikeThis, methodNamesLikeThis, CONSTANT_VALUES_LIKE_THIS, foo.namespaceNamesLikeThis.bar, and filenameslikethis.js.
2. Comments using JSDoc
    - The following is a sample of JSDoc:

```
/**
 * A JSDoc comment should begin with a slash and 2 asterisks.
 * Inline tags should be enclosed in braces like {@code this}.
 * @desc Block tags should always start on their own line.
 */
```

3. Strings prefer single-quotes over double-quotes
    - For consistency single-quotes (') are preferred to double-quotes ("). This is helpful when creating strings that include HTML.

As a way to maintain style and code consistency, we used *eslint*, a linting tool for Javascript to analyse and check our code. A `.eslintrc.json` file is included in the root of the Zoomable directory, which is used to configure the rules for our code.

5 TESTING

All the frontend and backend tests are packed into a Grunt Task called 'test'. To run all predefined test cases, you can type:

```
$ grunt test
```

5.1 Backend Testing

Mocha and Supertest are used to test our backend APIs. For each controller file, there will be corresponding Mocha test cases. For example, if the controller file is named `VideoController.js`, there will be a corresponding testing file called `VideoController.test.js`. For each API, there should be at least one test case to check if the API is implemented correctly.

To run the backend test independently, you can type:

```
$ npm test
```

5.2 Frontend Testing

Jasmine is used to test our frontend functions, with Karma as our test runner. The Karma configuration file, `karma.config.js`, is added in the root of Zoomable directory. This configuration file specifies the various options for karma, including the list of dependencies and files to load when running the unit tests. These tests will run on PhantomJS in the terminal, and can be called to run independently (from the backend test) using the following command:

```
$ karma start
```

For each controller, directive, and service files, there will be corresponding unit test cases written using Jasmine. For example, if the controller file is named `loginController.js`, a corresponding test file called `loginController.spec.js` can be found in the `test/frontend` folder.

5.3 Continuous Integration

The Continuous Integration (CI) service of our choice is Travis CI. To link Travis CI to the Zoomable repository, we need to login to Travis CI with our GitHub account, accepting the GitHub access permissions confirmation. Then we enable Travis CI for Zoomable GitHub

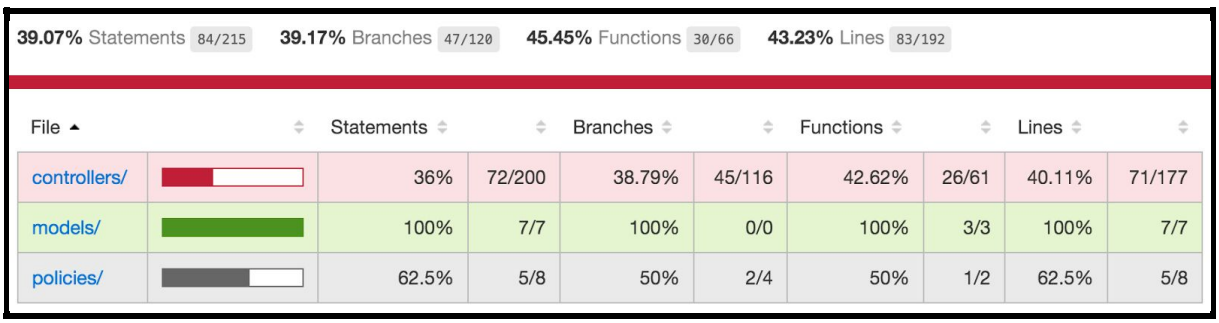
repository. In order to tell Travis CI what to build, a `.travis.yml` file is added to the repository. The following is our `.travis.yml` file:

```
language: node_js

node_js:
  - "0.10"

before_install: npm install -g grunt-cli

install:
  - npm install
  - bower install

script: grunt test

branches:
  only:
    - develop
    - master
```

Currently, Travis CI is only running our test cases on Node version 0.10 and on develop and master branches.

5.4 Code Coverage

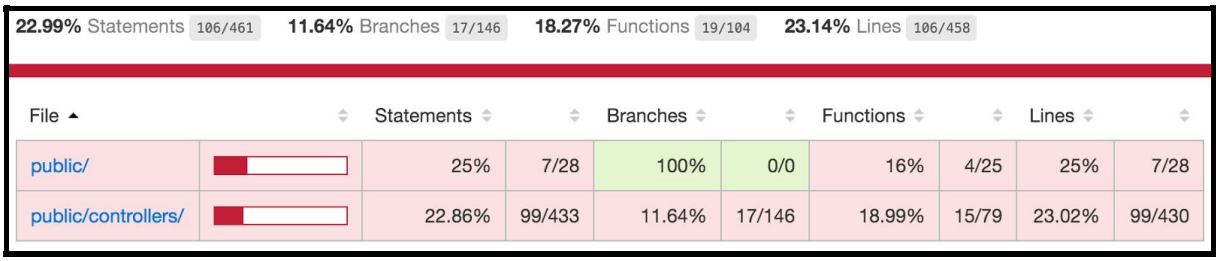
A new `coverage` folder will be generated from running the tests through grunt. It will contain the code coverage reports for frontend and backend, added into the `client` and `server` folder respectively. The screenshots below illustrates the current status of our test coverage as reported in the code coverage reports.



39.07% Statements 84/215 **39.17%** Branches 47/120 **45.45%** Functions 30/66 **43.23%** Lines 83/192

File	Statements		Branches		Functions		Lines	
controllers/	36%	72/200	38.79%	45/116	42.62%	26/61	40.11%	71/177
models/	100%	7/7	100%	0/0	100%	3/3	100%	7/7
policies/	62.5%	5/8	50%	2/4	50%	1/2	62.5%	5/8

Figure 5.4.a. Code Coverage Report for Backend Tests



File	Statements		Branches		Functions		Lines	
public/	25%	7/28	100%	0/0	16%	4/25	25%	7/28
public/controllers/	22.86%	99/433	11.64%	17/146	18.99%	15/79	23.02%	99/430

Figure 5.4.b. Code Coverage Report for Frontend Tests

6 FUTURE DEVELOPMENT AND TECHNICAL ISSUES

6.1 Technical Issues

In this section, we provide the details on the technical issues we encountered in the developing of Zoomable with the attempted solutions.

6.1.1 Video Player

Browser Compatibility

Our Video Player is built on top of Shaka Player's library for video streaming. As we have been developing only on Google Chrome, Zoomable currently does not work well with other browsers such as FireFox and Safari. This is due to browser differences in how HTML5 Canvas behaves as well as frame-accurate seeking. A possible solution would be to remove the dependency on Shaka Player by using Broadway or a similar solution for decoding, so as to decouple the streaming and decoding functions currently provided by Shaka.

Synchronisation

One of the major problems faced was in terms of seeking a specific frame for all the videos, where we could not ensure that all tiles were showing the correct frame at any point in time, so that the video picture would be synchronised (i.e. no tiling line visible due to minor differences in frames). Despite seeking to the same current time for every video, each video would show different frames for a given time. We attempted to use video.js library for frame-accurate seeking, but it did not solve this problem. We also attempted to remove B-frames during the video's encoding. We found out that these did not work due to an accuracy issue with video seeking in Chrome[1], and that we should be able to seek to a single frame accurately in other browsers. A possible solution is to enable compatibility with other browsers.

6.2 Future Development

Firstly and foremostly, future development entails completing all the requirements as listed in Section 2. These include user experience improvements such as being able to ensure the uploaded videos are not corrupt, and setting a maximum recommended zoom level in the UI for optimal viewing experience.

[1] https://bugs.chromium.org/p/chromium/issues/detail?id=66631

Secondly, to further improve on the player, selective decoding could be implemented to reduce the amount of extra resources used to stream the tiles that are not in view when the video is zoomed in. Resolving current issues, such as synchronization and browser compatibility, are also key to the intended user experience.

Thirdly, video statistics can be enhanced to include more viewing interactions, such as the total duration of the video that is watched by individual viewers, or certain sections of the video that are repeatedly watched by viewers. This will help to provide a more complete statistical analysis of the engagement rate of a particular video, which could be useful for marketing purposes.

7 REPOSITORY

The repository for Zoomable is located at: https://github.com/nus-mtp/zoomable.js. The structure of Zoomable is as follows, along with a description for each folder and file:

`> api`	Folder containing backend implementation code for controllers, models, policies, responses and services
`> assets`	Folder containing frontend implementation code for application and video player, along with styles and images to display on the frontend
`> config`	Folder containing configuration files for backend
`> coverage`	Folder containing code coverage reports for backend and frontend tests
`> libs`	Folder containing external libraries and dependencies used in frontend, generated from `bower.json` file
`> node_modules`	Folder containing all the NPM modules, specified in `package.json`
`> scripts`	Folder containing the bash script that automate the processing of the uploaded video
`> tasks`	Folder containing all the Grunt Tasks and Configuration files (Task Automation)
`> test`	Folder containing the frontend and backend test cases
`> views`	Folder containing *.ejs* template files for frontend interface
`app.js`	File that tell Node how to start the app when not running `sails`

	`lift`
`Gruntfile.js`	Grunt file for asset management
`karma.config.js`	Configuration file for Karma to run frontend unit tests
`bower.json`	Manifest file specifying the list of external libraries and dependencies for frontend
`package.json`	Standard configuration file for NPM
`LICENSE.md`	License file for Zoomable.
`README.md`	Generic README to describe the application
`Dockerfile`	File to automate building of image using Docker

Appendix A.1: User stories

1. As a Content Manager, I can upload video into the video server so I can play it from the video list.
2. As a Content Manager, I can delete videos that I have uploaded.
3. As a Content Manager, I can list of videos that I have uploaded.
4. As a Content Manager, I can sort my list of uploaded videos based on popularity (most viewed), privacy mode (public, unlisted) and/or date added (oldest, newest).
5. As a Content Manager, I can search for a particular video that I have uploaded.
6. As a Content Manager, I can embed an uploaded video in a video player into another website.
7. As a Content Manager, I can edit the information of an uploaded video. (Title, Description, Tags, Privacy)
8. As a Content Manager, I can add subtitles for a video.
9. As a Content Manager, I can see the number of views for each of the uploaded video.
10. As a Content Manager, I can see a heatmap of the most zoomed regions for each of the uploaded video.
11. As a Viewer, I can play and pause a video.
12. As a Viewer, I can replay the video from the beginning.
13. As a Viewer, I can see time elapsed of the video that is playing.
14. As a Viewer, I can put my video on fullscreen.
15. As a Viewer, I can zoom in (and out) of a video so that I can view a certain part of it in greater detail, without loss of quality to the video.
16. As a Viewer, I can pan around the zoomed in video, so that I can view other parts of the video in greater detail.
17. As a Viewer, I can see where I am zooming in to in the video with a small map of the entire video.
18. As a Viewer, I can choose to play the video from any point in time of the video through the seek bar, so that I can play from a certain segment of a video.
19. As a Viewer, I can adjust the volume of the video, or mute the audio if I want.
20. As a Viewer, I can share the video to anyone.
21. As a Viewer, I can see a small thumbnail over the seek bar to look through snapshot of a certain frame in the video.
22. As a Viewer, I can take a snapshot of the specific frame and download it as an image.
23. As a Viewer, I can show or hide subtitle for a video if it is available.

24. As a Viewer, I can add annotations to the seek bar to indicate a point-of-interest at a specific time of the video, so that the user can have a better overview of the video and can jump straight to it.
25. As a Viewer, I can use keyboard shortcuts to control zoom and pan direction.
26. As a Viewer, I can adjust the speed of the video for playback.

Appendix A.2: Abuser stories

1. As a hacker, I want to try all possible passwords using brute force attack to gain unauthorised access to the dashboard as a content manager so I can have access to the videos.
2. As a hacker, I want to get unauthorised access into the database using code injection.

Appendix A.3: Use cases

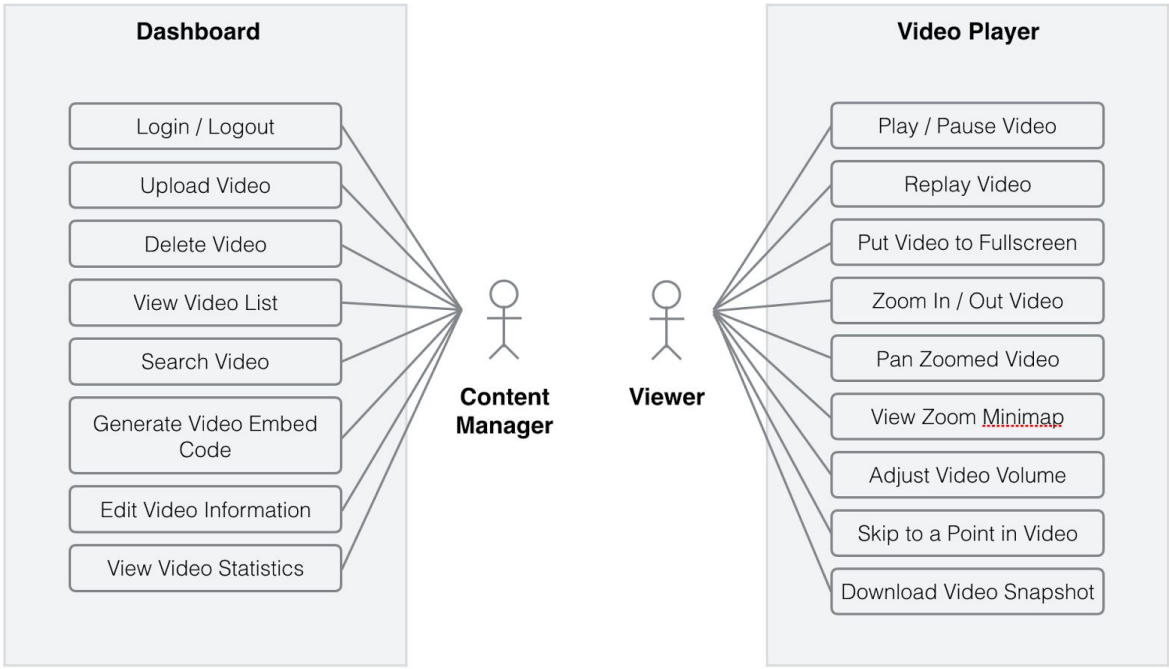


Figure A.3.a. Use Case Diagram for Zoomable

An overview of the use cases showing how the actors interact with the system is illustrated in the figure above. There are two main actors and two main systems. Content Manager is the actor in the Dashboard system, while Viewer is the actor in the Video Player system. A complete list of the use cases are detailed below.

Clarification of terms:

- Precondition: What the system will ensure is true before letting the use case start, generally indicates that some other use case has already run to set it up
- Extensions: What the video server can detect and handle

Use Case #1: Login to Dashboard

System: Dashboard

Actor: Content Manager

Precondition: A Content Manager account has already been created.

Success Guarantees: Content Manager will be logged into Dashboard.

Main Success Scenario:

1. Dashboard requests for username and password.

2. Content Manager keys in username and password.
3. Content Manager is logged into Dashboard.
   Use case ends.

Extensions:

3a. Login credentials are incorrect.

3ai. Dashboard prompts Content Manager to enter login credentials again.

Use case resumes from step 3 if login is successful.

3b. Login credentials are incorrect for the third time consecutively.

3bi. Dashboard activates reCAPTCHA to verify if Content Manager is human or robot.

Use case resumes from step 3 if login is successful.

Use case ends.

Use Case #2: Upload a Video

System: Dashboard

Actor: Content Manager

Precondition: Content Manager is logged into Dashboard.

Success Guarantees: Content Manager will have video uploaded on the Video Server.

Main Success Scenario:

1. Content Manager chooses to upload video.
2. Dashboard requests for video file.
3. Content Manager selects a video file.
4. Dashboard uploads the selected video.
   Use case ends.

Extensions:

4a. Selected video is in an unsupported format

4ai. Dashboard prompts Content Manager to select a video in a supported format.

4aii. Content Manager selects another video.

Use case resumes from step 4.

4b. Content Manager chooses to cancel the upload.

4bi. Dashboard requests to confirm the cancellation.

4bii. Content Manager confirms the cancellation.

Use case ends.

4c. Network connection to the Video Server is lost.

4ci. Dashboard requests Content Manager to upload video again.

Use case resumes from step 3.

Use Case #3: Delete a Video

System: Dashboard

Actor: Content Manager

Precondition: Content Manager is already logged into the dashboard. There is video in the Video Server for deletion.

Success Guarantees: Video will be deleted from the Video Server.

Main Success Scenario:

1. Content Manager deletes a video from Dashboard.
2. Dashboard requests Content Manager to confirm the deletion of video.
3. Content Manager confirms the deletion of video.
4. Dashboard deletes the video.

Use case ends.

Extensions:

3a. Content Manager chooses to cancel the deletion of video.

3ai. Dashboard cancels the deletion of video.

Use case ends.

Use Case #4: Sort Video List in Dashboard by Popularity

System: Dashboard

Actor: Content Manager

Precondition: Content Manager is already logged in to the Dashboard. There are videos in the Video Server.

Success Guarantees: Content Manager can see a list of sorted videos.

Main Success Scenario:

1. Content Manager chooses to sort videos by popularity in Dashboard.
2. Dashboard displays a list of sorted videos by popularity.

Use case ends.

Use Case #5: Search For Video By Title in Dashboard

System: Dashboard

Actor: Content Manager

Precondition: Content Manager is already logged into the Dashboard. There are videos in the Video Server.

Success Guarantees: Content Manager can search for video by title.

Main Success Scenario:

1. Content Manager keys in video title.
2. Dashboard displays a list of videos with relevant titles.
   Use case ends.

Extensions:

2a. No video title matches the title keyed by Content Manager.
   2ai. Dashboard informs Content Manager there is no match.
   Use case ends.

Use Case #6: Generate Video Embed Code

System: Dashboard

Actor: Content Manager

Precondition: Video Player plays a video.

Success Guarantees: Content Manager can generate an embedded link from the Dashboard.

Main Success Scenario:

1. Content Manager selects a video and selects *Embedded Link*.
2. Dashboard displays embed code for the selected video.
   Use case ends.

Use Case #7: Edit Information of a Video

System: Dashboard

Actor: Content Manager

Precondition: Content Manager is already logged into the Dashboard.

Success Guarantees: All changes made to the video information will be saved.

Main Success Scenario:

1. Content Manager chooses to edit a video.
2. Dashboard redirects Content Manager to *Edit Video* page.
3. Content Manager edits information of the video and saves the changes.
4. Dashboard saves the changes made by Content Manager.

Use case ends.

Use Case #8: Add Video Subtitles

System: Dashboard

Actor: Content Manager

Precondition: Content Manager is already logged in to Dashboard and at *Edit Video* page.

Success Guarantees: Video subtitles will be added to the video.

Main Success Scenario:

1. Content Manager chooses to add subtitles to a video.
2. Dashboard requests subtitle file for the video.
3. Content Manager selects a subtitle file.
4. Dashboard uploads the subtitle file.

   Use case ends.

Extensions:

4a. Subtitle file is in an unsupported format.

   4ai. Dashboard prompts Content Manager to select a subtitle file in a supported format.

   4aii. Content Manager selects another subtitle file.

   Use case resumes from step 4.

Use Case #9: Review View-Related Statistics

System: Dashboard

Actor: Content Manager

Precondition: Content Manager is already logged into the Dashboard. There are videos in the Video Server.

Success Guarantees: Content Manager can review view-related statistics pertaining to video.

Main Success Scenario:

1. Content Manager selects a video from Dashboard.
2. Dashboard redirects content manager to *Edit Video* page.
3. Content Manager selects option to review view-related statistics.
4. Dashboard displays view-related statistics of video.

   Use case ends.

Use Case #10: Play a Video

System: Video Player

Actor: Viewer

Precondition: There is a video in the Video Server.

Success Guarantees: Viewer can start video or continue from where they last paused.

Main Success Scenario:

1. Viewer plays a video.
2. Video Player plays the video until it ends.
   Use case ends.

Extensions:

2a. Network connection to the Video Server is lost.
   2ai. Video Player continues to play the video until buffer runs out.
   2aii. Video Player informs viewer that the network connection is lost .
   Use case ends.

Use Case #11: Pause a Video

System: Video Player

Actor: Viewer

Precondition: A video is playing in the Video Player.

Success Guarantees: Viewer can stop their video at the last-displayed point.

Main Success Scenario:

1. Viewer chooses to pause the video.
2. Video Player pauses the video.
   Use case ends.

Use Case #12: Replay a Video

System: Video Player

Actor: Viewer

Precondition: A video has finished playing in the Video Player.

Success Guarantees: The video will play from the start.

Main Success Scenario:

1. Viewer replays the video.
2. Video Player will play the video from the beginning.
   Use case ends.

Extensions:

2a. Network connection to the Video Server is lost.

2ai. Video Player informs Viewer that the network connection is lost.

Use case ends.

Use Case #13: Put Video on Fullscreen Mode

System: Video Player

Actor: Viewer

Precondition: Video Player is not in *Fullscreen* mode.

Success Guarantees: Video Player will expand to fit the device screen.

Main Success Scenario:

1. Viewer chooses to view the video in *Fullscreen* mode.
2. Video Player expands to fit the device screen completely.

Use case ends.

Extensions:

2a. Viewer chooses to exit *Fullscreen* mode.

2ai. Video Player exits *Fullscreen* mode.

Use case ends.

Use Case #14: Zooming In on a Video

System: Video Player

Actor: Viewer

Precondition: There is a video is playing in the Video Player.

Success Guarantees: Video will be zoomed in according to Viewer's zoom-level preferences.

Main Success Scenario:

1. Viewer zooms in to the video.
2. Video Player displays a zoomed in view of the video.

Use case ends.

Extensions:

1a. The video is already zoomed in to its maximum magnification.

Use case ends.

Use Case #15: Zooming Out on a Video

System: Video Player

Actor: Viewer

Precondition: There is a video is playing in the Video Player.

Success Guarantees: Video will be zoomed out according to Viewer's zoom-level preferences.

Main Success Scenario:

1. Viewer zooms out of the video.
2. Video Player displays a zoomed out view of the video.

   Use case ends.

Extensions:

1a. The video is already zoomed out to the minimum magnification.

   Use case ends.

Use Case #16: Panning Around a Zoomed In Video

System: Video Player

Actor: Viewer

Precondition: There is a zoomed in video is playing in the Video Player.

Success Guarantees: Other regions of the zoomed in video will be shown.

Main Success Scenario:

1. Viewer pans around the zoomed in video.
2. Video Player displays the video accordingly to the Viewer's panned area.

   Use case ends.

Use Case #17: Using Minimap Reference

System: Video Player

Actor: Viewer

Precondition: There is a zoomed in video is playing in the Video Player.

Success Guarantees: Viewer can use the minimap to take reference of the position of the video being zoomed in.

Main Success Scenario:

1. Viewer zooms in to the video.
2. Video Player displays a minimap to indicate the current position of the zoomed in video.
3. Viewer pans to the left of the video.
4. Video Player displays the change in position in the minimap.

Use Case #18: Skip to a Point of a Video

System: Video Player

Actor: Viewer

Precondition: There is a video is playing in the Video Player.

Success Guarantees: The video will jump to a certain point in time of the video.

Main Success Scenario:

1. Viewer selects a certain point on the seek bar.
2. Video Player starts playing the video from the point of the seek bar selected by Viewer.

   Use case ends.

Use Case #19: Increase Volume of Video

System: Video Player

Actor: Viewer

Precondition: Video Player volume is not at maximum volume level.

Success Guarantees: Volume of audio increased according to Viewer's preference.

Main Success Scenario:

1. Viewer chooses to increase the volume of the audio.
2. Video Player increases volume of the audio accordingly.

   Use case ends.

Extensions:

1a. Audio is at maximum volume.

   Use case ends.

Use Case #20: Decrease Volume of Video

System: Video Player

Actor: Viewer

Precondition: Video Player volume is not at minimum volume level.

Success Guarantees: Volume of audio decreased according to user's input.

Main Success Scenario:

1. Viewer chooses to decrease the volume of the audio.
2. Video Player decreases volume of the audio accordingly.

   Use case ends.

Extensions:

1a. Audio is at minimum volume.

   Use case ends.

Use Case #21: Share Video

System: Video Player

Actor: Viewer

Success Guarantees: Viewer shares the video via a link.

Main Success Scenario:

1. Viewer chooses to share the video.
2. Video Player displays a link for the Viewer to share the video.
   Use case ends.

Use Case #22: View Thumbnail at a Point on Seek Bar

System: Video Player

Actor: Viewer

Precondition: There is a video is playing in the Video Player and the video has been buffered.

Success Guarantees: Viewer can view a thumbnail of the video's frame at that point on the seek bar.

Main Success Scenario:

1. Viewer hovers the mouse cursor over a point on the seek bar.
2. Video Player displays a thumbnail of the frame at the point where cursor is hovered.
   Use case ends.

Use Case #23: Download Video Snapshot

System: Video Player

Actor: Viewer

Precondition: There is a video playing in the Video Player.

Success Guarantees: Viewer downloads a snapshot of the video.

Main Success Scenario:

1. Viewer chooses to download a snapshot of the video when the video is playing in the Video Player.
2. Video Player downloads a snapshot of the video.
   Use case ends.

Use Case #24: View Video Subtitle

System: Video Player

Actor: Viewer

Precondition: There are video subtitles for the video.

Success Guarantees: Viewer views the video subtitles.

Main Success Scenario:

1. Viewer chooses to show subtitles for the video.

2. Video Player displays the video subtitles.
   Use case ends.

Appendix A.4: Misuse Cases

Misuse Case #1: Get Unauthorised Access into Dashboard using Brute Force Attack

System: Dashboard

Actor: Hacker

Precondition: Hacker has access to Dashboard's URL.

Success Guarantees: reCAPTCHA will be shown to deter robot's brute force attack.

Main Success Scenario:

1. Hacker uses brute force attack by writing a script(robot) to cycle through all possible username and passwords to gain access to the dashboard as a content manager.
2. Dashboard temporarily disables content manager account after detecting incorrect login credentials for more than five times with reCAPTCHA.

Extensions:

1a. Login credentials are incorrect.
    1a1. Dashboard prompts hacker to enter login credentials again.
    Use case resumes from step 1.

1b. Login credentials are incorrect for the third time consecutively.
    1b1. Dashboard activates reCAPTCHA to verify if login is submitted by human or robot.
        Use case resumes from step 1.

Misuse Case #2: Get Unauthorised Access into Database using Code Injection

System: Video Server, Dashboard

Actor: Hacker

Success Guarantees: Content Manager input will always be validated before being written into the database.

Main Success Scenario:

1. Hacker manually submits a form with a longer value.
2. Dashboard validates the length of the input and detects that it is over the maximum value allowed.
3. Dashboard prompts an error and does not allow hacker to proceed to the next step.
   Use case ends.