SAIVT WebDemo | Final Report

Francesco Ferraioli n8323143

TABLE OF CONTENTS

1. PROJECT OUTLINE

Speech Audio Image Video Techologies (SAIVT) has been researching and implementing numerous techniques and algorithms for entity extraction from video's. Entities can be faces shown on the video, voices that are heard, objects and many more. Recently SAIVT has decided to showcase all of these research findings and algorithm through a web site. The project that I undertook involved various improvements and feature implementation and enhancements to this web site.

The first task I undertook was implementing enhancements to the website in terms of the User Interface (UI). I had been informed that the front end of the application is not written following best practices. First and foremost, my task was to change this and ensure that the code for the front end is written well, easy to understand and renders as intended. Furthermore, once that was complete and the front end of the application was written well, I began to make the UI more user friendly and nice to look at. This will involved changing the style and the look and feel of the pages.

Moreover, there was some functionality in place for editing tags for a video. However, the functionality that was in place was not very user friendly and required the user to follow specific rules. Part of this task was to change that to a more user friendly implementation, allowing the user to simply remove them and add them individually and very easily.

The second and probably largest, most difficult and time demanding task was implementing the functionality to add video's through the website itself. In this task a user would be able to navigate to a page to upload a video of their own to the website.

Once the user had filled in the various information related to the video and then selected the video to upload and submitted the form, the server would then be able to receive this data. Once the data is received, the server would then store the metadata temporarily and send the video off to be processed by a High Performance Computer (HPC) and extract the various entities. Once the processing is done, the entities extracted would be clustered with the current entities in the database from the other videos. What that means is checking if an entity that has been extracted from the video is the same as an entity extracted in another.

The third and final task of the project was to add enhancements to the displaying of the entity extraction results on the video play page. Once a face has been detected by the system during the processing, information is stored about the whereabouts on the video the face is shown. The system then creates another video from that video but adds on top a hollow square around the face, a different color for each face entity it sees on that video. This new video is the video that is displayed on the video play page. Part of this task was be to make the face on the video interactive and that once the user hovers over it, it will show information about that entity.

Another enhancement to that page was to show a label on the bottom left hand of the video with the current speaker and a small image of them. This label would also be video interactive and that once the user hovers over it, it will show information about that entity.

2. PROJECT OUTCOME

The project outline listed various tasks for the entity extraction webdemo project from SAIVT. This section will outline my achievements on those tasks. The tasks are listened below:

1. User Interface Enhancements
2. Add Video Feature
3. Entity Display Enhancements

I will now discuss my progress on these tasks individually. Inside each section I will have a further two subsections, one explaining the functionality and one explaining the implementation. The functionality will delve into what exactly the outcome was whereas the implementation will discuss how this was implemented, containing some code snippets from different files.

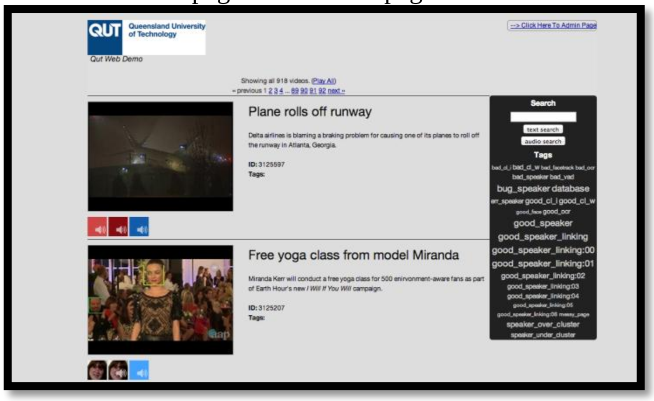
2.1. USER INTERFACE ENHANCEMENTS

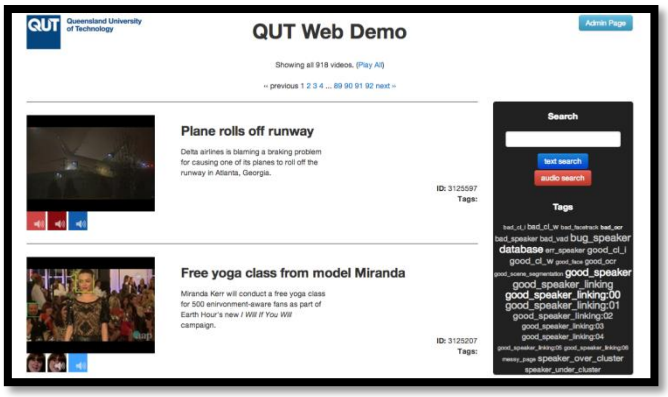
2.1.1. FUNCTIONALITY

The two main purpose's of this task were firstly to ensure that the code for the front end is written well, easy to understand and renders as intended and secondly to make the UI more user friendly and pleasing to look at.

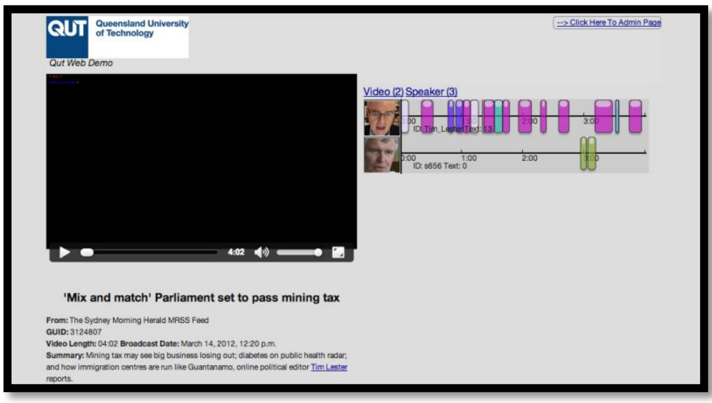
I started off by taking a lot of the inline styling out of elements into classes and put those classes into a separate css file which I included in the file header. This made the html much easier to read and manage and furthermore implementing best practices. Moreover, the base html file was written very poorly, having various blocks for each page to implement, whereas it should simply contain two blocks, a head block and a content block and the html which should remain constant for all the pages that inherit from it. This allows for all those page to only have to specify two blocks making it very clear to the developer how the page is going to render in a browser.

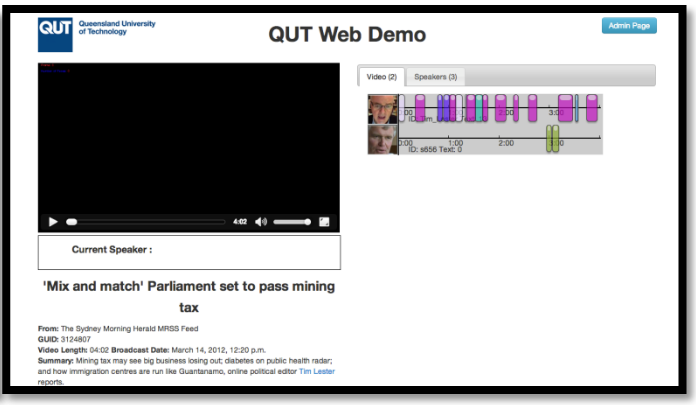
The next step was to introduce the Bootstrap Framework for further enhancements to the UI. Bootstrap has a very nice layout grid template using divs and I used this to style the elements where I wanted them to be, taking away a lot of the previously unmanageable styling which was used to place elements on the page. Bootstrap also has a very nice list of classes to style buttons and I used these to my advantage to make the page much more pleasant to look at from a user perspective. The images below shows the old look compared with the new look of the index page which is the page which lists out videos.


Queensland University of Technology
Qut Web Demo
--> Click Here To Admin Page
Showing all 918 videos. (Play All)
« previous 1 2 3 4 ... 89 90 91 92 next »
Plane rolls off runway
Free yoga class from model Miranda
Search
text search
audio search
Tags


Queensland University of Technology
QUT Web Demo
Admin Page
Showing all 918 videos. (Play All)
« previous 1 2 3 4 ... 89 90 91 92 next »
Plane rolls off runway
Free yoga class from model Miranda
Search
text search
audio search
Tags

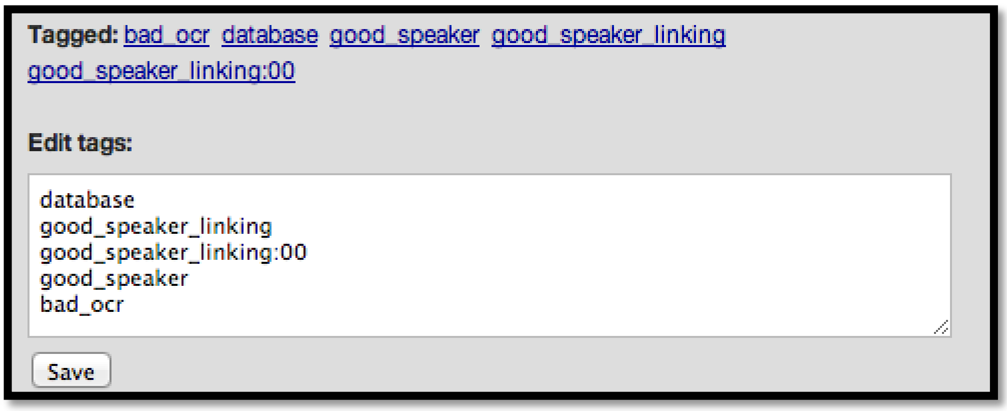
When I finished styling the index page I moved on to the video play page which is the page that renders when a particular video has been chosen by the user to play and to see what entities belong to this particular video.


Queensland University of Technology
Qut Web Demo
--> Click Here To Admin Page
Video (2) Speaker (3)
'Mix and match' Parliament set to pass mining tax


Queensland University of Technology
QUT Web Demo
Admin Page
Video (2)
Speakers (3)
Current Speaker :
'Mix and match' Parliament set to pass mining tax

The above images compare the old and the new look of the video play page. The main styling differences are the added tabs on the right hand side showing the Faces and the Speakers. Previously to switch between Faces and Speakers the user needed to click the links above and that would hide and show the respective entities. Now the Faces and Speakers are separated in tabs and thus it's a simple as clicking the tabs to switch between the entities. The tabs are much nicer to look at than the individual links that were previously available.

The last major enhancement to the UI and user friendliness of the web pages is the way we are currently allowing the user to edit the tags of a video. Previously this was done through the use of a textarea where all the tags were put on the textarea, each tag on one line. If a user wished to edit them they would need to edit them straight on the textarea and ensure that they are kept each in one line and there aren't any spaces as there is no validation. The image below shows how the editing of tags was previously done.

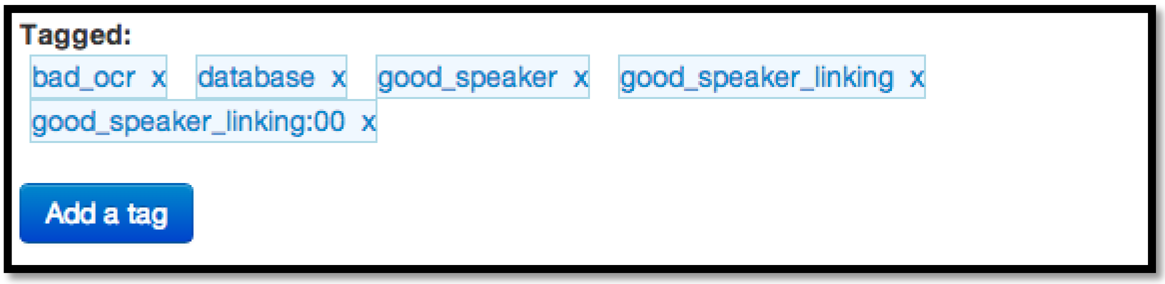


Tagged: bad_ocr database good_speaker good_speaker_linking good_speaker_linking:00

Edit tags:

```
database
good_speaker_linking
good_speaker_linking:00
good_speaker
bad_ocr
```

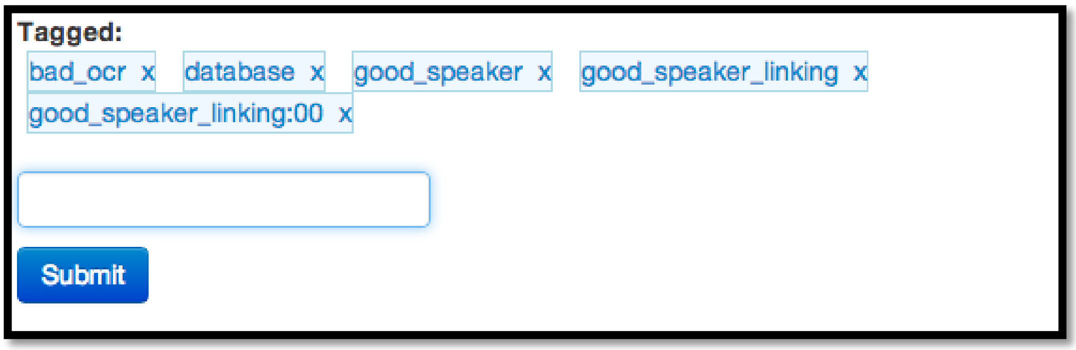
The new method of editing tags that I implemented is very different. All the tags are listed out with a cross next to them that when clicked will delete that tag. Below the list is a "Add a tag" button, this is all shown below.



Tagged:

bad_ocr x database x good_speaker x good_speaker_linking x good_speaker_linking:00 x

Clicking the button will reveal a textbox to enter the new tag and a button to submit.

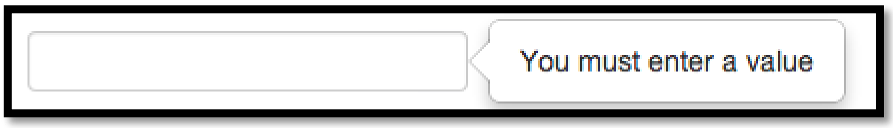


Tagged:

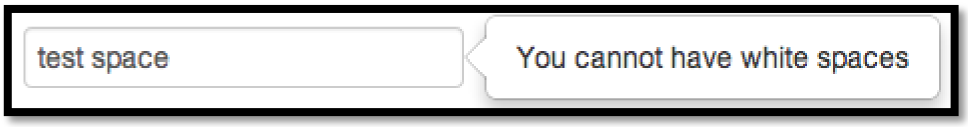
bad_ocr x database x good_speaker x good_speaker_linking x good_speaker_linking:00 x

Unlike the previous method, this method has validation to ensure a good tag is added to the list. Validation includes:

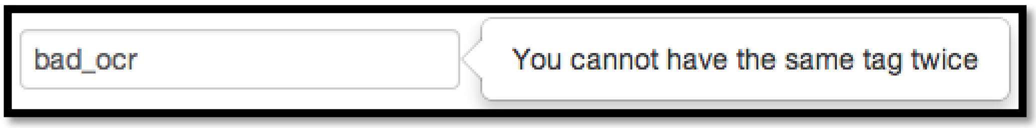
1. Presence



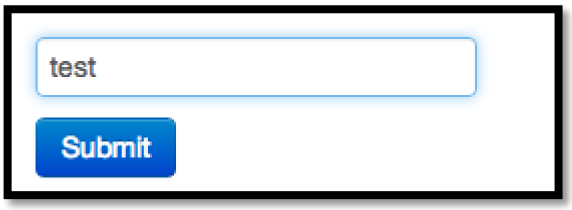
2. No spaces

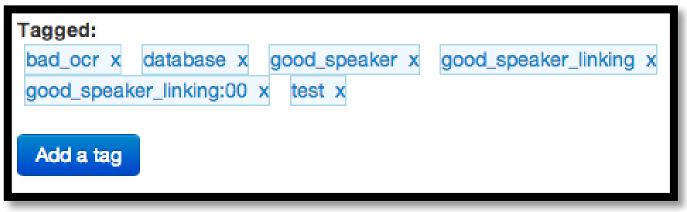


3. No duplicates



Once an input from the user is given that passes all these validation tests, it is added to the list.

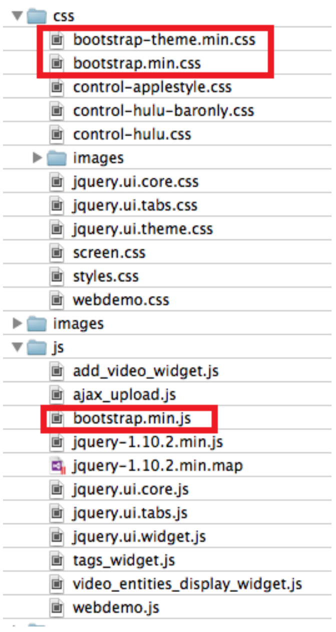




This is all achieved via ajax, so no reload to the page is necessary. This allows for a very enjoyable and pleasant user experience.

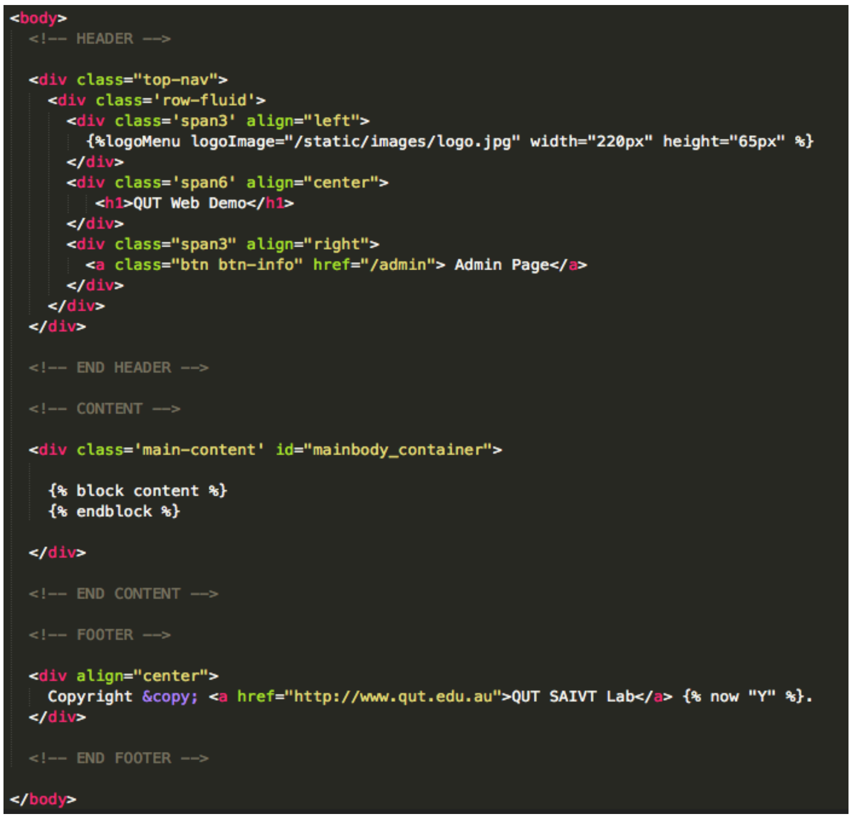
2.1.2. IMPLEMENTATION

As explained, a major part of this task involved integrating the Bootstrap framework into the webdemo. This included adding the css bootstrap files as well as the js bootstrap file.



```
▼ css
    bootstrap-theme.min.css
    bootstrap.min.css
    control-applestyle.css
    control-hulu-baronly.css
    control-hulu.css
  ▶ images
    jquery.ui.core.css
    jquery.ui.tabs.css
    jquery.ui.theme.css
    screen.css
    styles.css
    webdemo.css
▶ images
▼ js
    add_video_widget.js
    ajax_upload.js
    bootstrap.min.js
    jquery-1.10.2.min.js
    jquery-1.10.2.min.map
    jquery.ui.core.js
    jquery.ui.tabs.js
    jquery.ui.widget.js
    tags_widget.js
    video_entities_display_widget.js
    webdemo.js
```

After adding the files, it was necessary to start using the classes provided by bootstrap. I started rewriting the structure of the HTML base page to be structured more professionally using the bootstrap classes. The following image displays the html of the base page in which every page in the webdemo inherits from.



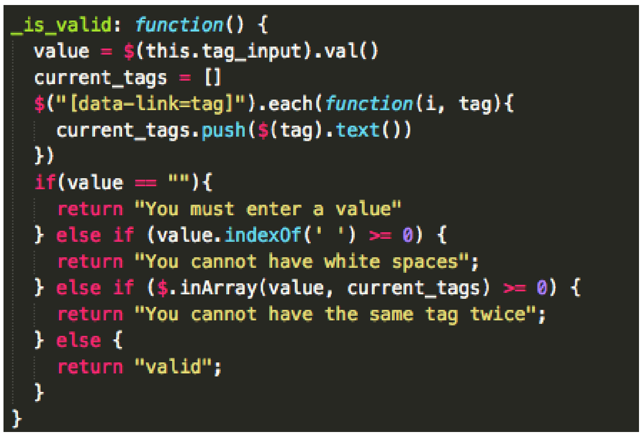
```html
<body>
  <!-- HEADER -->

  <div class="top-nav">
    <div class='row-fluid'>
      <div class='span3' align="left">
        {%logoMenu logoImage="/static/images/logo.jpg" width="220px" height="65px" %}
      </div>
      <div class='span6' align="center">
         <h1>QUT Web Demo</h1>
      </div>
      <div class="span3" align="right">
        <a class="btn btn-info" href="/admin"> Admin Page</a>
      </div>
    </div>
  </div>

  <!-- END HEADER -->

  <!-- CONTENT -->

  <div class='main-content' id="mainbody_container">

    {% block content %}
    {% endblock %}

  </div>

  <!-- END CONTENT -->

  <!-- FOOTER -->

  <div align="center">
    Copyright &copy; <a href="http://www.qut.edu.au">QUT SAIVT Lab</a> {% now "Y" %}.
  </div>

  <!-- END FOOTER -->

</body>
```

Each page that inherits from this base page will need to have a content block which will be placed in the content area of the base page. The header and footer and constant and will remain the same for every page.

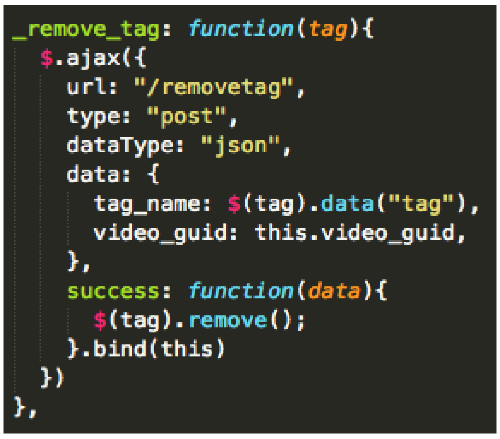
The changing from links to tab for the right hand side of the video play page was done using the tabs widget from jquery.

The tags functionality was all implemented using a widget I wrote. The containing file is tags_widget.js in the js folder of the media folder.

This file contains the javascript to run the functionality. It includes the validation code and ajax calls.



```
_is_valid: function() {
  value = $(this.tag_input).val()
  current_tags = []
  $("[data-link=tag]").each(function(i, tag){
    current_tags.push($(tag).text())
  })
  if(value == ""){
    return "You must enter a value"
  } else if (value.indexOf(' ') >= 0) {
    return "You cannot have white spaces";
  } else if ($.inArray(value, current_tags) >= 0) {
    return "You cannot have the same tag twice";
  } else {
    return "valid";
  }
}
```

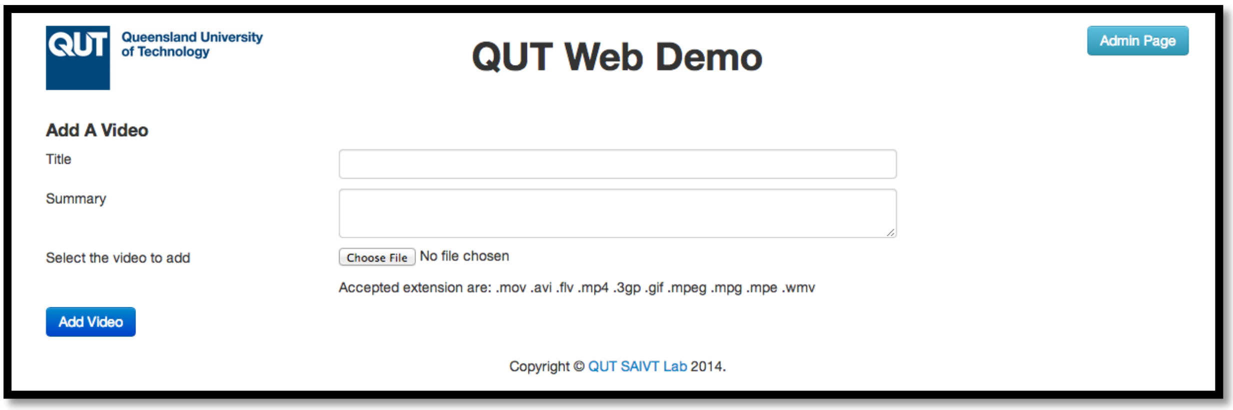


```
_remove_tag: function(tag){
  $.ajax({
    url: "/removetag",
    type: "post",
    dataType: "json",
    data: {
      tag_name: $(tag).data("tag"),
      video_guid: this.video_guid,
    },
    success: function(data){
      $(tag).remove();
    }.bind(this)
  })
},
```

2.2. ADD VIDEO FEATURE

2.2.1. FUNCTIONALITY

The next task that I tackled was that of implementing the "Add Video" feature. I started off by creating a page that contains the form to add a video, the page is displayed below.



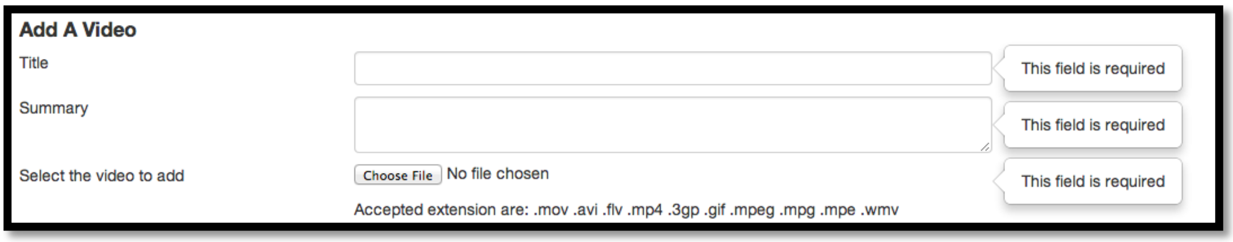
Add A Video

Title	
Summary	
Select the video to add	Choose File No file chosen
	Accepted extension are: .mov .avi .flv .mp4 .3gp .gif .mpeg .mpg .mpe .wmv

Copyright © QUT SAIVT Lab 2014.

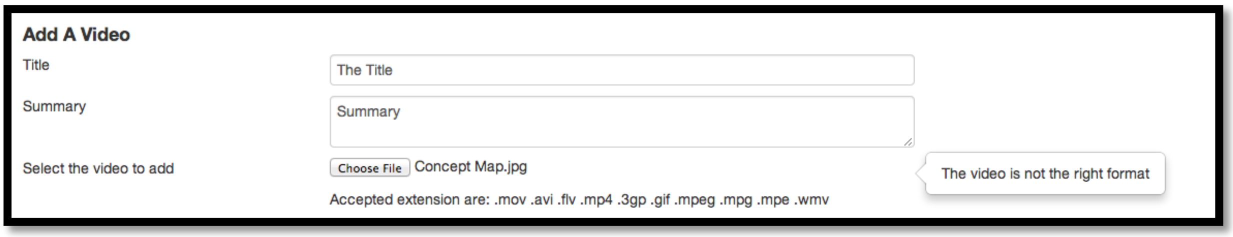
In this page the user can enter the title and the summary they wish to have for the video as well as choosing the video itself. Once the user has finished filling out the form they click the Add Video button to submit the form. This form has different validation requirements that must be met for the video to be uploaded.

1. Presence


Add A Video

Title		This field is required
Summary		This field is required
Select the video to add	Choose File No file chosen	This field is required
	Accepted extension are: .mov .avi .flv .mp4 .3gp .gif .mpeg .mpg .mpe .wmv	

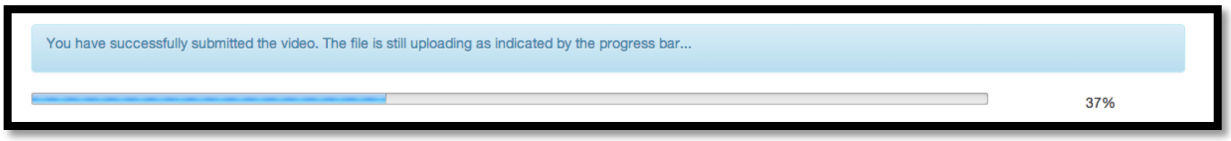
2. File Format


Add A Video

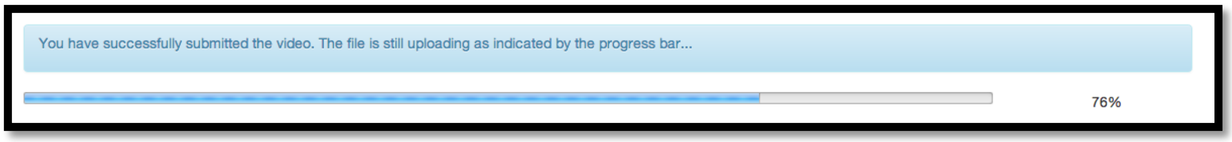
Title	The Title	
Summary	Summary	
Select the video to add	Choose File Concept Map.jpg	The video is not the right format
	Accepted extension are: .mov .avi .flv .mp4 .3gp .gif .mpeg .mpg .mpe .wmv	

The user is able to choose any file they wish but the form will not submit unless the file format matches one of the formats listed there. The list of accepted extensions are kept in one centralized place so that changing the list will change the html and the validation at the same time, making it very easy to add and remove accepted extensions.

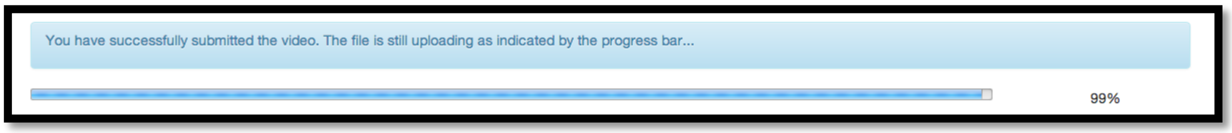
Once a user successfully passes the validation of the form the form is submitted and the video starts uploading. Depending on the size of the file it may take longer to upload. I have implemented a progress bar that shows once the form is submitted. The progress bar shows how much of the file is uploaded to the user. This ensure that the user is kept notified of the progress and also enhances user experience. Below are screenshots taken of the progress bar at different stages.


You have successfully submitted the video. The file is still uploading as indicated by the progress bar...

37%

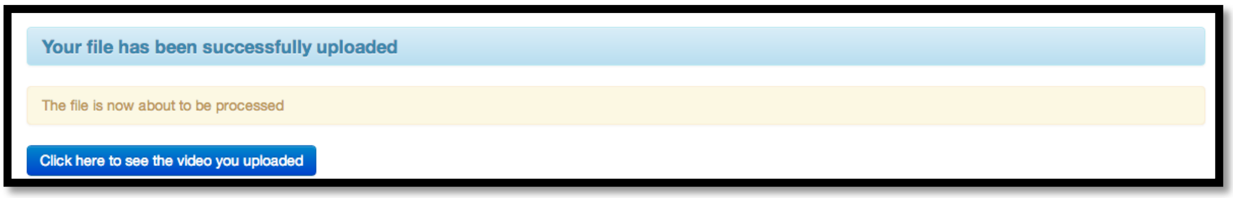

You have successfully submitted the video. The file is still uploading as indicated by the progress bar...

76%


You have successfully submitted the video. The file is still uploading as indicated by the progress bar...

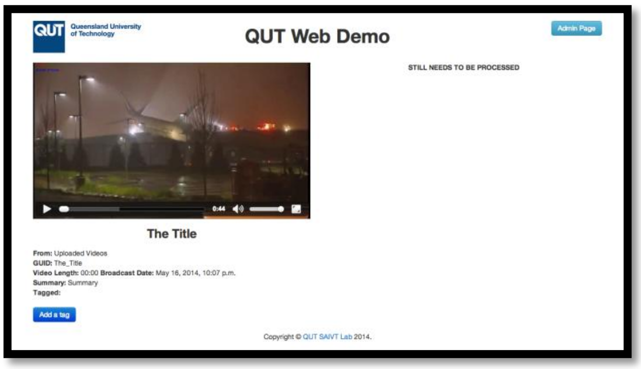
99%

Once the upload is complete, the user is notified and a button is presented to them to take them to the page which will show them the newly uploaded video.


Your file has been successfully uploaded

The file is now about to be processed

The video however has just been uploaded and has not gone through any processing for entity extraction. A video that has not been processed yet will have the processed flag not set and the user will be shown if the video has been processed or not. The image below shows what the user is shown when they click the button. The page they are shown is the video play page with the video that was just uploaded and the text to show that it has not yet been processed. The video is also present on the index page. Again, on this shows that the video is still to be processed.



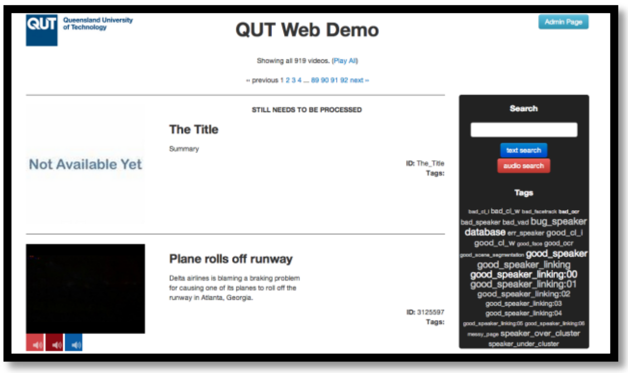
QUT Web Demo

STILL NEEDS TO BE PROCESSED

The Title

From: Uploaded Videos
GUID: The_Title
Video Length: 00:00 **Broadcast Date:** May 16, 2014, 10:07 p.m.
Summary: Summary
Tagged:

Copyright © QUT SAIVT Lab 2014.


Queensland University of Technology
Admin Page
Search
text search
audio search
Tags
bad_cl_i bad_cl_w bad_facetrack bad_ocr bad_speaker bad_vad bug_speaker database err_speaker good_cl_i good_cl_w good_face good_ocr good_scene_segmentation good_speaker good_speaker_linking good_speaker_linking:00 good_speaker_linking:01 good_speaker_linking:02 good_speaker_linking:03 good_speaker_linking:04 good_speaker_linking:05 good_speaker_linking:06 messy_page speaker_over_cluster speaker_under_cluster

QUT Web Demo

Showing all 919 videos. (Play All)

« previous 1 2 3 4 ... 89 90 91 92 next »

STILL NEEDS TO BE PROCESSED

Not Available Yet

The Title

Summary

ID: The_Title
Tags:

Plane rolls off runway

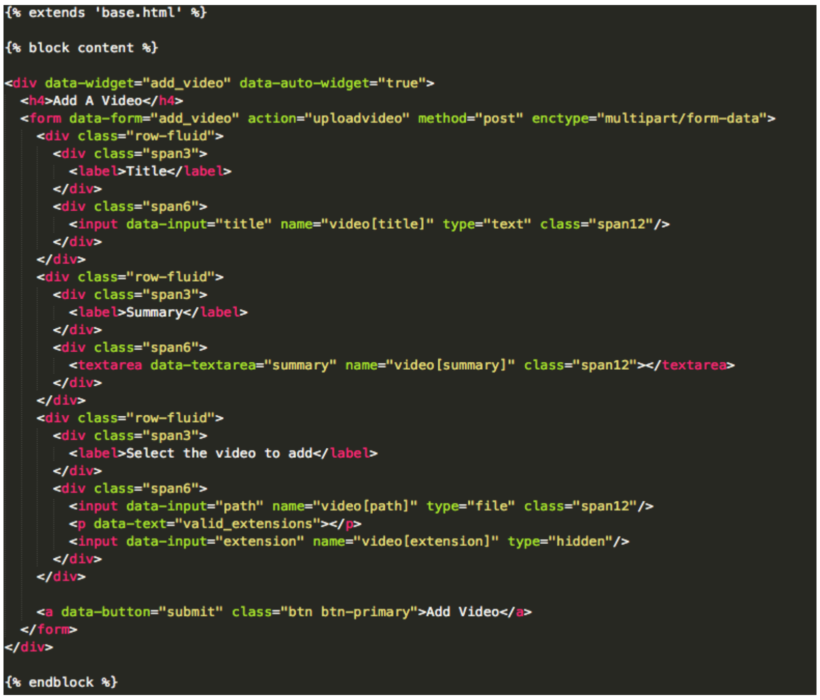
Delta airlines is blaming a braking problem for causing one of its planes to roll off the runway in Atlanta, Georgia.

ID: 3125597
Tags:

Now that the uploaded video is saved into the database and shown to the user. I then set up a cron job that would then send the video to a High Performance Computer (HPC) to process. I tried to include the starting of the processing on the HPC via the cron job but that turned out to be probelematic and we decided that it was best to, for now, let it be initiated manually. Upon completion of the processing, the video along with all its extracted entities would be pulled back to the webdemo server and the database would be rebuilt with the new information about all the entities found for the video uploaded. This is all done via the one cron job that runs every 30 minutes.

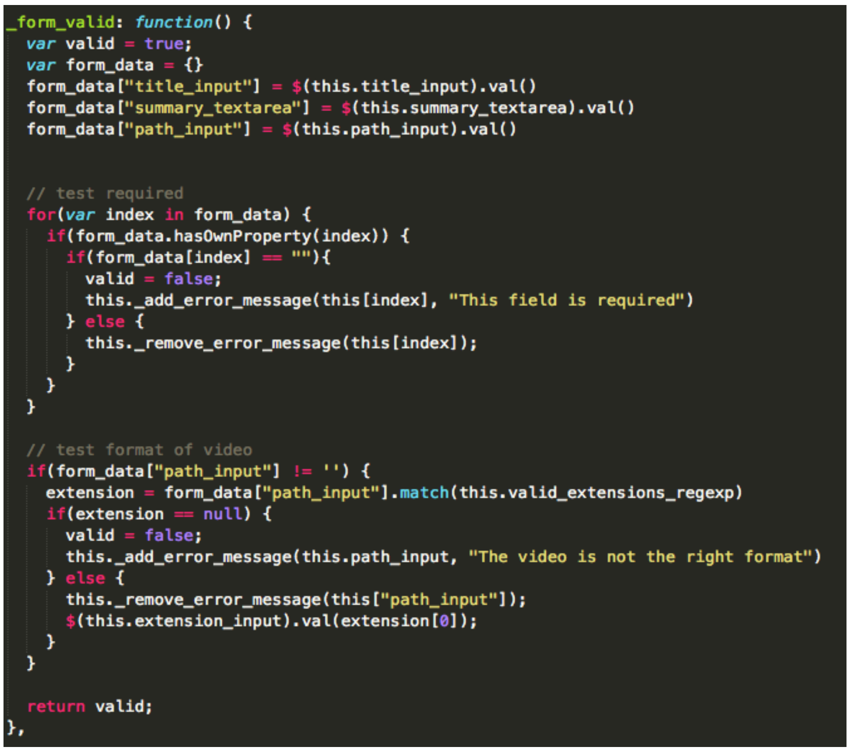
2.2.2. IMPLEMENTATION

The add video form HTML is contained in addvideo.html. The content of the file is simply a form and the content is shown in the below snippet.



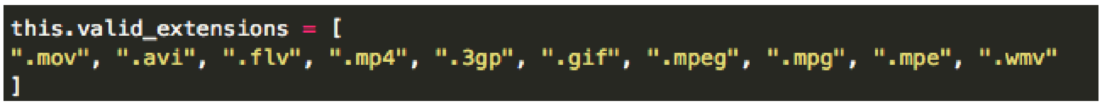
```
{% extends 'base.html' %}

{% block content %}

<div data-widget="add_video" data-auto-widget="true">
  <h4>Add A Video</h4>
  <form data-form="add_video" action="uploadvideo" method="post" enctype="multipart/form-data">
    <div class="row-fluid">
      <div class="span3">
        <label>Title</label>
      </div>
      <div class="span6">
        <input data-input="title" name="video[title]" type="text" class="span12"/>
      </div>
    </div>
    <div class="row-fluid">
      <div class="span3">
        <label>Summary</label>
      </div>
      <div class="span6">
        <textarea data-textarea="summary" name="video[summary]" class="span12"></textarea>
      </div>
    </div>
    <div class="row-fluid">
      <div class="span3">
        <label>Select the video to add</label>
      </div>
      <div class="span6">
        <input data-input="path" name="video[path]" type="file" class="span12"/>
        <p data-text="valid_extensions"></p>
        <input data-input="extension" name="video[extension]" type="hidden"/>
      </div>
    </div>

    <a data-button="submit" class="btn btn-primary">Add Video</a>
  </form>
</div>

{% endblock %}
```

The javascript that runs the page is found in a widget on its own. The widget is called the add_video and is contained in the add_video_widget.js file. This jquery widget contains a lot of functionality, including the validation code.



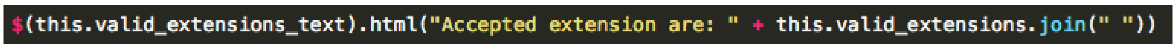
```
_form_valid: function() {
  var valid = true;
  var form_data = {}
  form_data["title_input"] = $(this.title_input).val()
  form_data["summary_textarea"] = $(this.summary_textarea).val()
  form_data["path_input"] = $(this.path_input).val()


  // test required
  for(var index in form_data) {
    if(form_data.hasOwnProperty(index)) {
      if(form_data[index] == ""){
        valid = false;
        this._add_error_message(this[index], "This field is required")
      } else {
        this._remove_error_message(this[index]);
      }
    }
  }

  // test format of video
  if(form_data["path_input"] != '') {
    extension = form_data["path_input"].match(this.valid_extensions_regexp)
    if(extension == null) {
      valid = false;
      this._add_error_message(this.path_input, "The video is not the right format")
    } else {
      this._remove_error_message(this["path_input"]);
      $(this.extension_input).val(extension[0]);
    }
  }

  return valid;
},
```

The valid video extensions are also stored in a centralized place in the widget.



```
this.valid_extensions = [
".mov", ".avi", ".flv", ".mp4", ".3gp", ".gif", ".mpeg", ".mpg", ".mpe", ".wmv"
]
```

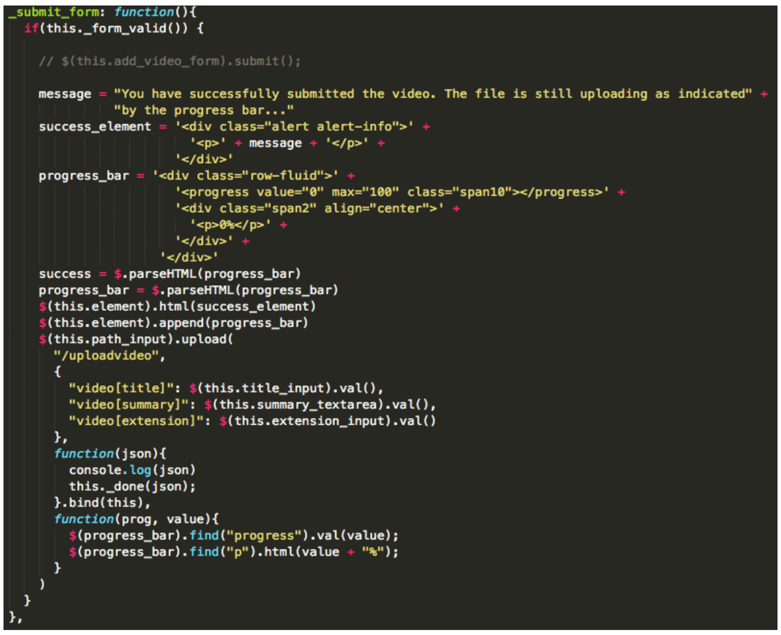
When the widget is loaded, the widget will go and add the text to the page informing the user of what the accepted extensions are for the video. This list is generated from the array stored in the widget.



```
$(this.valid_extensions_text).html("Accepted extension are: " + this.valid_extensions.join(" "))
```

The main functionality to run the progress bar is also stored in the widget as shown below, however, most of the functionality is stored in another file.

The following code is the code that is called when the submit button is pressed

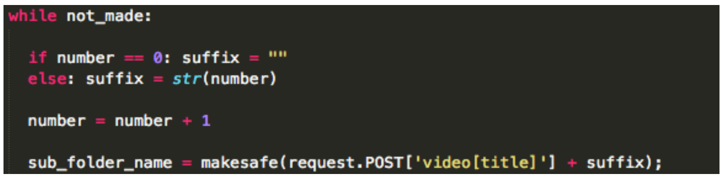


```
_submit_form: function(){
  if(this._form_valid()) {

    // $(this.add_video_form).submit();

    message = "You have successfully submitted the video. The file is still uploading as indicated" +
              "by the progress bar..."
    success_element = '<div class="alert alert-info">' +
                        '<p>' + message + '</p>' +
                      '</div>'
    progress_bar = '<div class="row-fluid">' +
                      '<progress value="0" max="100" class="span10"></progress>' +
                      '<div class="span2" align="center">' +
                        '<p>0%</p>' +
                      '</div>' +
                    '</div>'
    success = $.parseHTML(progress_bar)
    progress_bar = $.parseHTML(progress_bar)
    $(this.element).html(success_element)
    $(this.element).append(progress_bar)
    $(this.path_input).upload(
      "/uploadvideo",
      {
        "video[title]": $(this.title_input).val(),
        "video[summary]": $(this.summary_textarea).val(),
        "video[extension]": $(this.extension_input).val()
      },
      function(json){
        console.log(json)
        this._done(json);
      }.bind(this),
      function(prog, value){
        $(progress_bar).find("progress").val(value);
        $(progress_bar).find("p").html(value + "%");
      }
    )
  }
},
```

The widget calls the upload function, but this upload function is stored in a different file, the file is called ajax_upload.js.

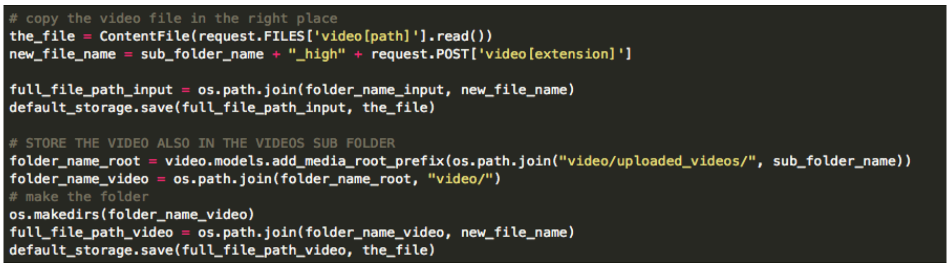
The form submits to an action in the view called uploadvideo found in views.py. This view action does many things:

1. Finds a name for the folder for the video ensuring no duplications. The folder name is based on the name of the video. If it finds a folder with the same name, it will add a suffix at the end of an incrementing number until it finds one that does not cause a duplicate.



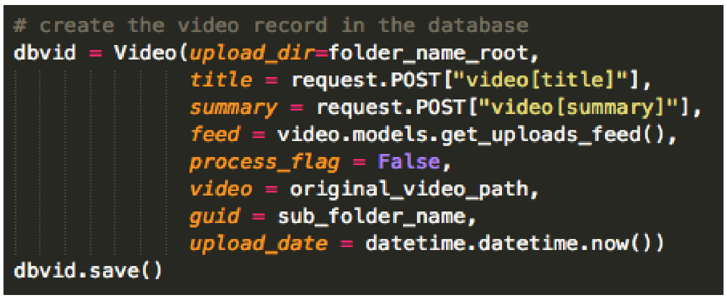
```
while not_made:

  if number == 0: suffix = ""
  else: suffix = str(number)

  number = number + 1

  sub_folder_name = makesafe(request.POST['video[title]'] + suffix);
```

2. Folder creation is made in two places. The first place is in media/uploaded_videos; this is the folder which will be synced across as the inputs to the processing program on lyra (the HPC). The second place is in media/videos/uploaded_videos; this is where the webdemo looks for videos to show on the website as uploaded videos. This is also where reloadvids looks for videos that are in the database.



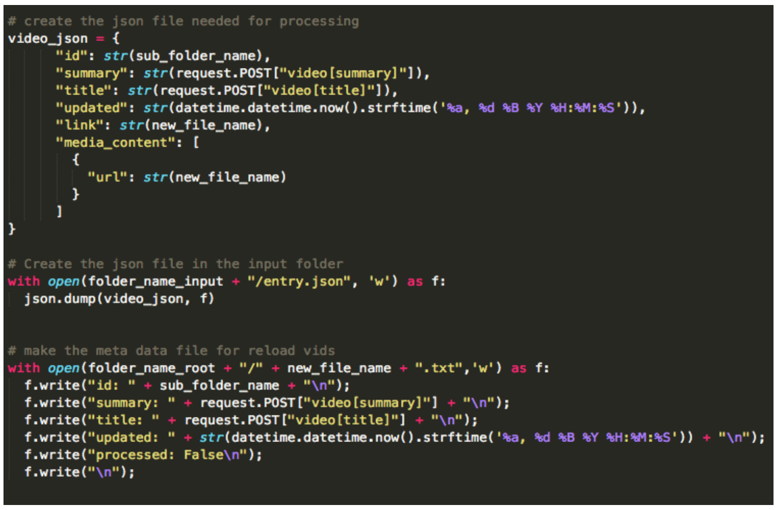
```python
# copy the video file in the right place
the_file = ContentFile(request.FILES['video[path]'].read())
new_file_name = sub_folder_name + "_high" + request.POST['video[extension]']

full_file_path_input = os.path.join(folder_name_input, new_file_name)
default_storage.save(full_file_path_input, the_file)

# STORE THE VIDEO ALSO IN THE VIDEOS SUB FOLDER
folder_name_root = video.models.add_media_root_prefix(os.path.join("video/uploaded_videos/", sub_folder_name))
folder_name_video = os.path.join(folder_name_root, "video/")
# make the folder
os.makedirs(folder_name_video)
full_file_path_video = os.path.join(folder_name_video, new_file_name)
default_storage.save(full_file_path_video, the_file)
```

3. It will then proceed to creating the video record in the database



```python
# create the video record in the database
dbvid = Video(upload_dir=folder_name_root,
              title = request.POST["video[title]"],
              summary = request.POST["video[summary]"],
              feed = video.models.get_uploads_feed(),
              process_flag = False,
              video = original_video_path,
              guid = sub_folder_name,
              upload_date = datetime.datetime.now())
dbvid.save()
```

4. Create the metadata files. The input to the processing program on lyra requires a json file along with the video that holds the metadata. On the other hand, reloadvids requires a txt file. The last step of the uploadvideo action is creating these two files in the right places.



```python
# create the json file needed for processing
video_json = {
      "id": str(sub_folder_name),
      "summary": str(request.POST["video[summary]"]),
      "title": str(request.POST["video[title]"]),
      "updated": str(datetime.datetime.now().strftime('%a, %d %B %Y %H:%M:%S')),
      "link": str(new_file_name),
      "media_content": [
        {
          "url": str(new_file_name)
        }
      ]
}

# Create the json file in the input folder
with open(folder_name_input + "/entry.json", 'w') as f:
  json.dump(video_json, f)


# make the meta data file for reload vids
with open(folder_name_root + "/" + new_file_name + ".txt",'w') as f:
  f.write("id: " + sub_folder_name + "\n");
  f.write("summary: " + request.POST["video[summary]"] + "\n");
  f.write("title: " + request.POST["video[title]"] + "\n");
  f.write("updated: " + str(datetime.datetime.now().strftime('%a, %d %B %Y %H:%M:%S')) + "\n");
  f.write("processed: False\n");
  f.write("\n");
```

As explained, once the video has been uploaded, a user will be able to view the video even if its yet to be processed as the video is in the database and the folder structure is correct for the webdemo to be able to display the video.

The cron job was the final thing that needed to be implemented for this task. The bash file the cron job needs to run is located in the webdemo folder, and it is called ReloadAndSyncUploaded.sh.

The bash file simply contains 3 commands.

1. Get the processed data from lyra's output folder

2. Run reloadvids

3. Send the uploaded videos to lyra's input folder

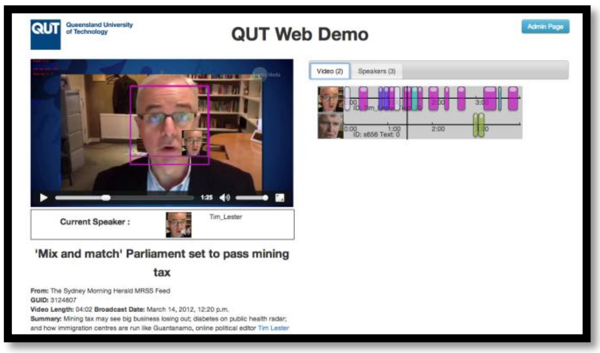
I also created a file called ReloadAndSyncUploadedHelp.txt. This file contains information about how to set up the cron job. It is important to note that for commands 1 and 3, it needs to ssh into lyra. Therefore public and private keys must be set up between the becks account and the lyra account for the cron job to work.

2.3. ENTITY DISPLAY ENHANCEMENTS

2.3.1. FUNCTIONALITY

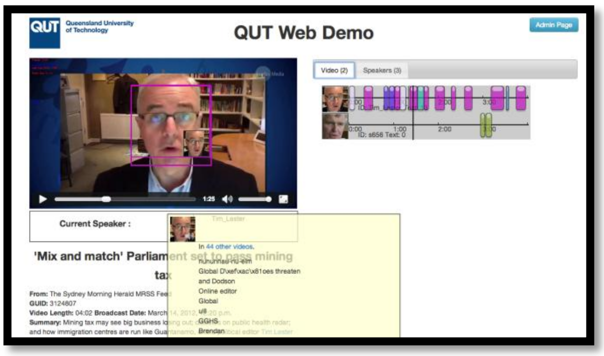
The next step was to implement the entity display enhancements. This task involves extracting the information regarding the current speaker and current faces and displaying it on the page. The system stores information that relates an entity to a time in seconds in a video. This information was needed to calculate who is the current speaker and who are the current faces at a particular point in time for a particular video that the user has decided to view. For the faces more information was necessary including the position of the face in the video. This information was stored and various calculations were done to determine where the face was relative to the particular web page.

The current speaker's image and id is displayed underneath the video. As the video plays, the current speaker is calculated from stored values and shown to the user. When a user hovers over the image, more information is displayed about the entity.

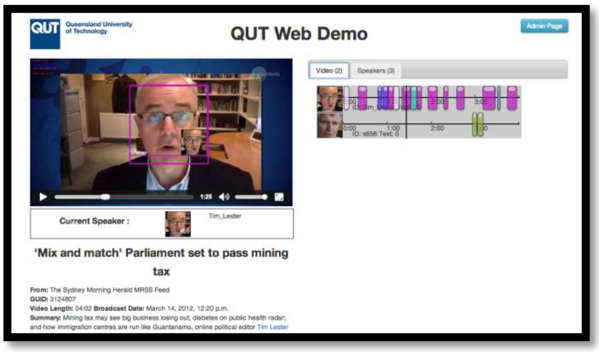


'Mix and match' Parliament set to pass mining tax

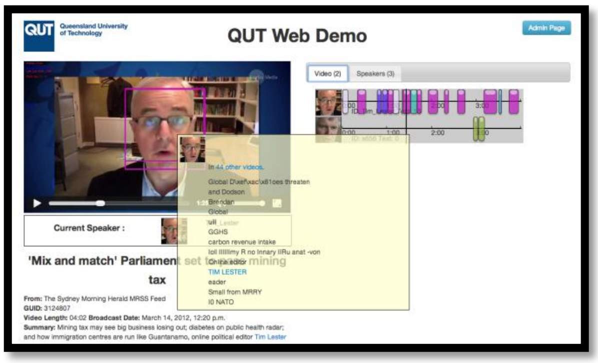
From: The Sydney Morning Herald MRSS Feed
GUID: 3124807
Video Length: 04:02 Broadcast Date: March 14, 2012, 12:20 p.m.
Summary: Mining tax may see big business losing out; diabetes on public health radar; and how immigration centres are run like Guantanamo, online political editor Tim Lester



Global D\xef\xac\x81oes threaten
and Dodson
Online editor
Global
GGHS
Brendan



On the other hand the current faces are displayed straight onto the video. A hollow rectangle is place on top of the video to show where on the video the face is shown. Calculations are made to determinate the position of the face on the video as well as the height and the width. On the bottom the box is the image of the current face and again, like with the speaker, once a user hovers over that image, more information is displayed about that entity.



Global D\xef\xac\x81oes threaten
and Dodson
Brendan
Global
GGHS
carbon revenue intake
China editor
TIM LESTER
eader
Small from MRRY
I0 NATO

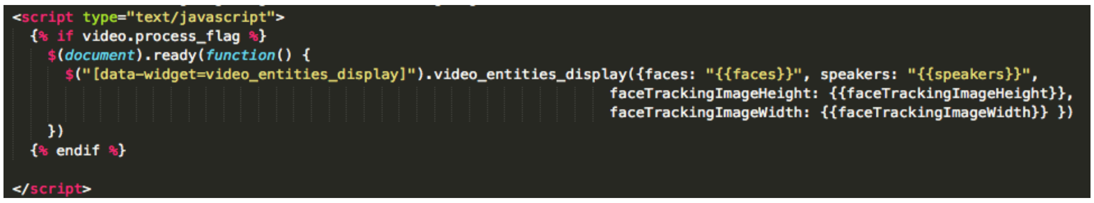
I then proceeded to replacing the encoded video to be the original video and having the overlaying HTML elements on top of the original video.

However, after implementing this feature, I found a bug in the storing of the x and y positions for the start and the end of the sections the faces appear. Appendix 1 describes this bug in a bug report.

Fixing the bug took sometime as I needed to familiarize myself with the face detection code, particularly the code that generates the files about the positioning of the faces. However, I managed to fix it in the end, changing the code to return the both the min and the max coordinates for each section. Having these figures allows the boxes that were drawn to be exactly the perfect size.

2.3.2. IMPLEMENTATION

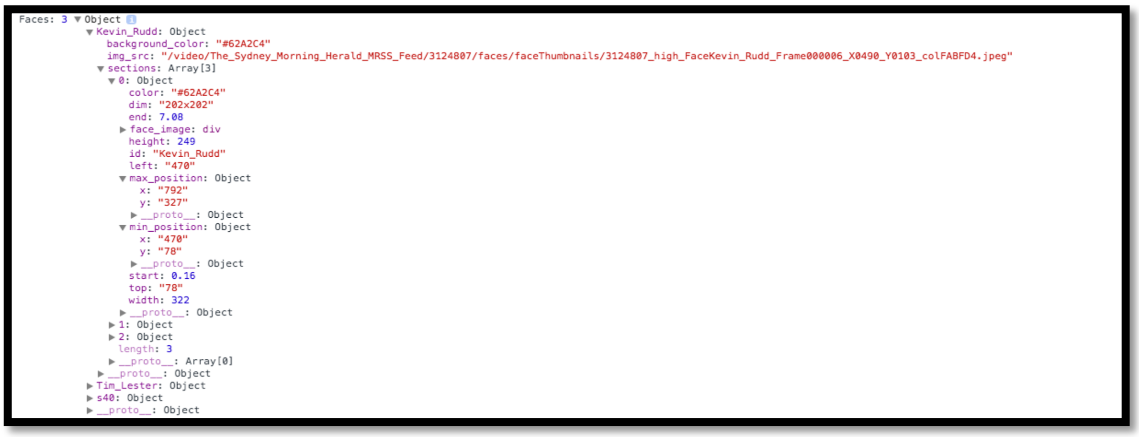
Most of the code to get this functionality to work is in javascript. Again I created a widget to encapsulate the javascript code. The server response to the videoplay page returns with the necessary details about the faces and the speakers for the particular video. A script is present in the HTML to load up the widget with the json data from the server response.



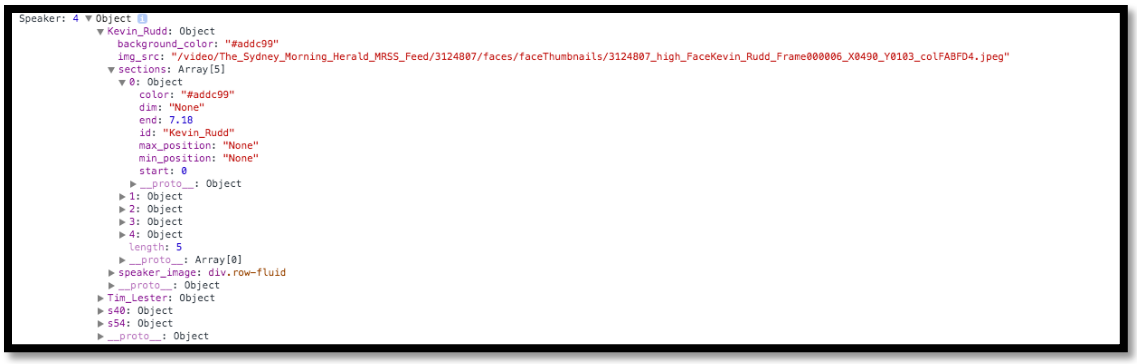
```
<script type="text/javascript">
  {% if video.process_flag %}
    $(document).ready(function() {
      $("[data-widget=video_entities_display]").video_entities_display({faces: "{{faces}}", speakers: "{{speakers}}",
                                                                        faceTrackingImageHeight: {{faceTrackingImageHeight}},
                                                                        faceTrackingImageWidth: {{faceTrackingImageWidth}} })
    })
  {% endif %}

</script>
```

The widget code is written in video_entities_display_widget.js. This file is fairly large and contains a lot of code to run the functionality of showing both the current speaker and the current faces.

The widget has a speakers object and a faces object. Both have very similar structure but one holds all the speakers for the video and one holds all the faces for the video. Each object holds an object for each entity, with the entities id as the key. Then each entity object has a sections array. This sections array holds all the times that an entity appears with the start time and end time. The faces entities objects is slightly more complex than the speakers as they need to hold also the positional values as well as the start and end times. A snippet of an example of both the speakers and the faces objects are present below.

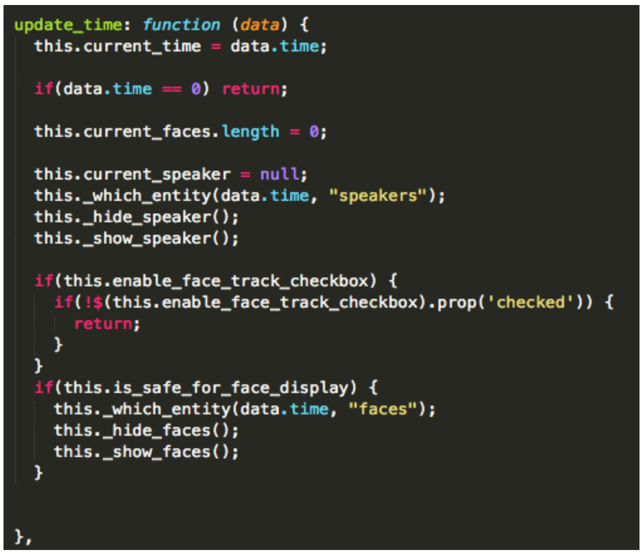


```
Faces: 3 ▼ Object
            ▼ Kevin_Rudd: Object
                background_color: "#62A2C4"
                img_src: "/video/The_Sydney_Morning_Herald_MRSS_Feed/3124807/faces/faceThumbnails/3124807_high_FaceKevin_Rudd_Frame000006_X0490_Y0103_colFABFD4.jpeg"
              ▼ sections: Array[3]
                ▼ 0: Object
                    color: "#62A2C4"
                    dim: "202x202"
                    end: 7.08
                  ▶ face_image: div
                    height: 249
                    id: "Kevin_Rudd"
                    left: "470"
                  ▼ max_position: Object
                      x: "792"
                      y: "327"
                    ▶ __proto__: Object
                  ▼ min_position: Object
                      x: "470"
                      y: "78"
                    ▶ __proto__: Object
                    start: 0.16
                    top: "78"
                    width: 322
                  ▶ __proto__: Object
                ▶ 1: Object
                ▶ 2: Object
                  length: 3
                ▶ __proto__: Array[0]
              ▶ __proto__: Object
            ▶ Tim_Lester: Object
            ▶ s40: Object
            ▶ __proto__: Object
```



```
Speaker: 4 ▼ Object
              ▼ Kevin_Rudd: Object
                  background_color: "#addc99"
                  img_src: "/video/The_Sydney_Morning_Herald_MRSS_Feed/3124807/faces/faceThumbnails/3124807_high_FaceKevin_Rudd_Frame000006_X0490_Y0103_colFABFD4.jpeg"
                ▼ sections: Array[5]
                  ▼ 0: Object
                      color: "#addc99"
                      dim: "None"
                      end: 7.18
                      id: "Kevin_Rudd"
                      max_position: "None"
                      min_position: "None"
                      start: 0
                    ▶ __proto__: Object
                  ▶ 1: Object
                  ▶ 2: Object
                  ▶ 3: Object
                  ▶ 4: Object
                    length: 5
                  ▶ __proto__: Array[0]
                ▶ speaker_image: div.row-fluid
                ▶ __proto__: Object
              ▶ Tim_Lester: Object
              ▶ s40: Object
              ▶ s54: Object
              ▶ __proto__: Object
```

The widget has a public function called update_time and it is shown below. This public function is called whenever the video time changes.



```
update_time: function (data) {
  this.current_time = data.time;

  if(data.time == 0) return;

  this.current_faces.length = 0;

  this.current_speaker = null;
  this._which_entity(data.time, "speakers");
  this._hide_speaker();
  this._show_speaker();

  if(this.enable_face_track_checkbox) {
    if(!$(this.enable_face_track_checkbox).prop('checked')) {
      return;
    }
  }
  if(this.is_safe_for_face_display) {
    this._which_entity(data.time, "faces");
    this._hide_faces();
    this._show_faces();
  }


},
```

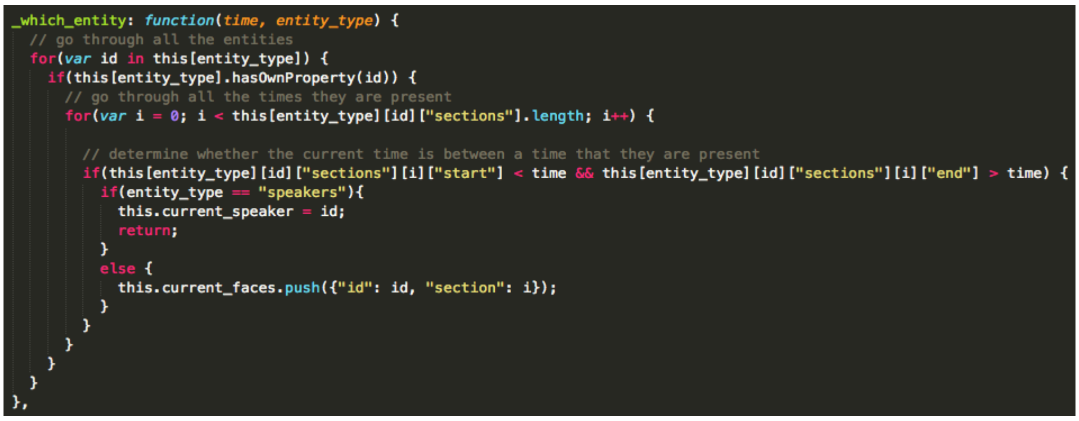
This function is rather straight forward.

1. Resets all the variables (sets the current speaker to null and clears the current faces array)

2. Checks which entities are showing

3. Hides and then shows the entities

The _which_entity function is pretty complex so I will explain that now.



```
_which_entity: function(time, entity_type) {
  // go through all the entities
  for(var id in this[entity_type]) {
    if(this[entity_type].hasOwnProperty(id)) {
      // go through all the times they are present
      for(var i = 0; i < this[entity_type][id]["sections"].length; i++) {

        // determine whether the current time is between a time that they are present
        if(this[entity_type][id]["sections"][i]["start"] < time && this[entity_type][id]["sections"][i]["end"] > time) {
          if(entity_type == "speakers"){
            this.current_speaker = id;
            return;
          }
          else {
            this.current_faces.push({"id": id, "section": i});
          }
        }
      }
    }
  }
},
```

The function goes through all the sections of all the entities of the video. If the start time is less than the current time of the video and the end time is greater than the current time of the video, it means that the current entity is active (be it a face on the video or a speaker in the audio). There can only be one speaker at a time, but there can be more than one face and this current faces is an array.

All the HTML elements are created on page load via the HTML template but are hidden. The widget will then find the correct one to show and hide all the other ones.

3. PROJECT TIMELINE

3.1. SEMESTER 1 - BEB801

		Project Week 1	Project Week 2	Project Week 3	Project Week 4	Project Week 5	Project Week 6	Project Week 7	Project Week 8	Project Week 9	Project Week 10	Project Week 11	Project Week 12	Project Week 13	Project Week 14
		24/02/14	03/03/14	10/03/14	17/03/14	24/03/14	31/03/14	07/04/14	14/04/14	21/04/14	28/04/14	05/05/14	12/05/14	19/05/14	26/05/14
Task	**Sub-Task**	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Mid-Sem Break	Week 9	Week 10	Week 11	Week 12	Week 13
	Clean up	🟦													
UI Enhancements	Bootstrap Integration		🟦	🟦											
	Tag CRUD				🟦										
	Page setup					🟦	🟦								
	Back-end setup							🟦	🟦						
Add Video	File transfer to Lyra														
	Processing														
	File transfer to server														
	Back-end setup														
Entity display enhancements	Current speaker label									🟦	🟦	🟦			
	Face tracking html element												🟦	🟦	🟦

3.2. SEMESTER 2 - BEB802

		Project Week 15	Project Week 16	Project Week 17	Project Week 18	Project Week 19	Project Week 20	Project Week 21	Project Week 22	Project Week 23	Project Week 24	Project Week 25	Project Week 26	Project Week 27	Project Week 28
		21/07/14	28/07/14	04/08/14	11/08/14	18/08/14	25/08/14	01/09/14	08/09/14	15/09/14	22/09/14	29/09/14	06/10/14	13/10/14	20/10/14
Task	**Sub-Task**	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Mid-Sem Break	Week 11	Week 12	Week 13
	Clean up														
UI Enhancements	Bootstrap Integration														
	Tag CRUD														
	Page setup														
	Back-end setup														
Add Video	File transfer to Lyra					🟦	🟦								
	Processing							🟦	🟦	🟦	🟦				
	File transfer to server											🟦	🟦		
	Back-end setup													🟦	🟦
Entity display enhancements	Current speaker label														
	Face tracking html element	🟦	🟦	🟦	🟦										

APPENDICIES

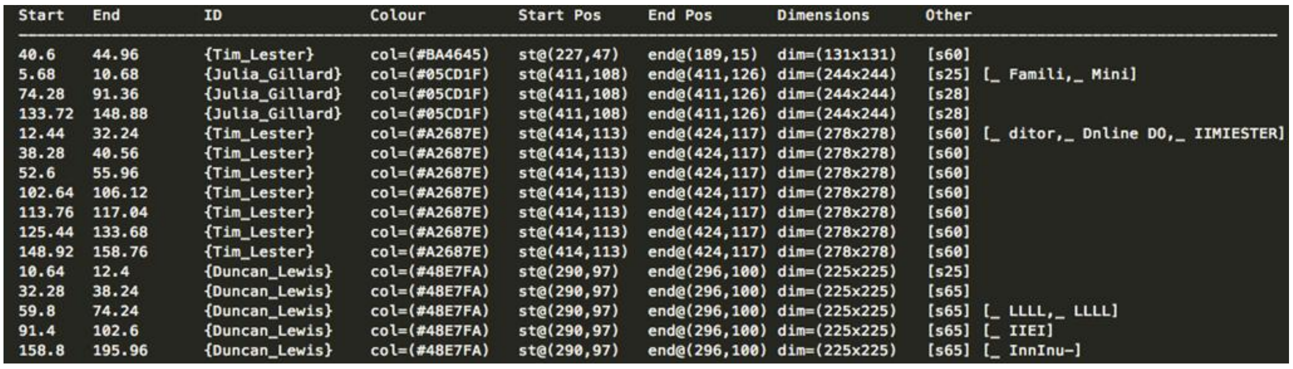
APPENDIX 1 - BUG REPORT

INTRODUCTION

One of my tasks was to implement a way to show the face tracking rectangle as an HTML element instead of it being part of the video. To do this, I required information about where exactly to place the element.

The entity extraction project currently stores a file which contains various information about the face tracking. The file contains the start and end time of when a particular face occurs. It contains the id of the entity as well as the colour the face tracking element is. Furthermore, it includes the start and end position as well as the dimension.

The content of a particular file is shown below.



```
Start   End      ID               Colour         Start Pos      End Pos        Dimensions       Other
------------------------------------------------------------------------------------------------------------
40.6    44.96    {Tim_Lester}     col=(#BA4645)  st@(227,47)    end@(189,15)   dim=(131x131)    [s60]
5.68    10.68    {Julia_Gillard}  col=(#05CD1F)  st@(411,108)   end@(411,126)  dim=(244x244)    [s25] [_ Famili,_ Mini]
74.28   91.36    {Julia_Gillard}  col=(#05CD1F)  st@(411,108)   end@(411,126)  dim=(244x244)    [s28]
133.72  148.88   {Julia_Gillard}  col=(#05CD1F)  st@(411,108)   end@(411,126)  dim=(244x244)    [s28]
12.44   32.24    {Tim_Lester}     col=(#A2687E)  st@(414,113)   end@(424,117)  dim=(278x278)    [s60] [_ ditor,_ Dnline DO,_ IIMIESTER]
38.28   40.56    {Tim_Lester}     col=(#A2687E)  st@(414,113)   end@(424,117)  dim=(278x278)    [s60]
52.6    55.96    {Tim_Lester}     col=(#A2687E)  st@(414,113)   end@(424,117)  dim=(278x278)    [s60]
102.64  106.12   {Tim_Lester}     col=(#A2687E)  st@(414,113)   end@(424,117)  dim=(278x278)    [s60]
113.76  117.04   {Tim_Lester}     col=(#A2687E)  st@(414,113)   end@(424,117)  dim=(278x278)    [s60]
125.44  133.68   {Tim_Lester}     col=(#A2687E)  st@(414,113)   end@(424,117)  dim=(278x278)    [s60]
148.92  158.76   {Tim_Lester}     col=(#A2687E)  st@(414,113)   end@(424,117)  dim=(278x278)    [s60]
10.64   12.4     {Duncan_Lewis}   col=(#48E7FA)  st@(290,97)    end@(296,100)  dim=(225x225)    [s25]
32.28   38.24    {Duncan_Lewis}   col=(#48E7FA)  st@(290,97)    end@(296,100)  dim=(225x225)    [s65]
59.8    74.24    {Duncan_Lewis}   col=(#48E7FA)  st@(290,97)    end@(296,100)  dim=(225x225)    [s65] [_ LLLL,_ LLLL]
91.4    102.6    {Duncan_Lewis}   col=(#48E7FA)  st@(290,97)    end@(296,100)  dim=(225x225)    [s65] [_ IIEI]
158.8   195.96   {Duncan_Lewis}   col=(#48E7FA)  st@(290,97)    end@(296,100)  dim=(225x225)    [s65] [_ InnInu-]
```

The guid of the video is 3085830.

The face detection program has picked up 3 different faces: Tim Lester, Julia Gillard and Duncan Lewis.

The file shows exactly when each face is detected along with the various information discussed earlier.

The important column to look at is the colour column, more than the ID. If you see, every row with the same colour has exactly the same start and end position, as well as the dimension.

The table is shown grouping the rows by colour, and this way the bug will be evident. It is clear that as one traverses down the table on any of the three columns (Start Position, End Position, Dimension) the value does not change until the colour value changes.

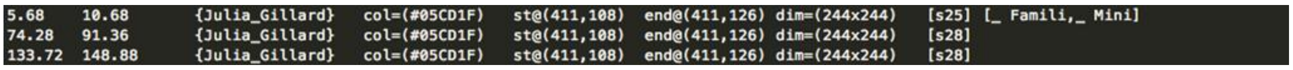
In the example shown above it is also clear that the issue is based on the colour not the ID. This is because the first row is a row containing information about Tim Lester with colour code #BA4645, whereas the 4th till the 10th record are also Tim Lester but with colour code of #A2687E and the values of the columns at question are different between the two colour codes even though the ID is the same.

To prove that this is not a coincidence and is in actual fact a bug, I could show more examples but instead I will stick with this example and look deeper into this video and see what the output of the face tracking program is.

BUG

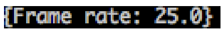
The video can be dived up into frames. Snapshots of the faces are taken per frame and the position values of the rectangle that surrounds the face are stored in the name of the thumbnail, along with the frame number. Using simple mathematics using the time in seconds the face appears and disappears as well as frames per seconds, it is easy to compute the corresponding frame number. This implies that it is possible to compare the values stored against the actual position values stored in the name of the thumbnail.

Analysing the three Julia Gillard rows that have the same colour code of #05CD1F, the bug can be easily exposed.



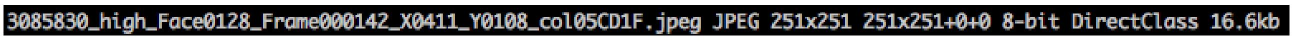
```
5.68     10.68       {Julia_Gillard}   col=(#05CD1F)   st@(411,108)  end@(411,126) dim=(244x244)   [s25] [_ Famili,_ Mini]
74.28    91.36       {Julia_Gillard}   col=(#05CD1F)   st@(411,108)  end@(411,126) dim=(244x244)   [s28]
133.72   148.88      {Julia_Gillard}   col=(#05CD1F)   st@(411,108)  end@(411,126) dim=(244x244)   [s28]
```

As previously stated, due to the bug, all the three rows have the same start and end position as well as the dimension, however, the thumbnail file names show that this is in fact incorrect.

Frame Rate: 

```
{Frame rate: 25.0}
```

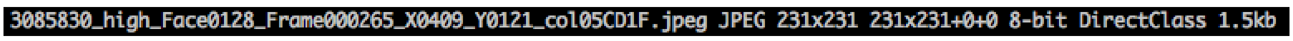
Row 1 states that the Julia Gillard's face first occurs at time 5.68 seconds into the video. Frames per seconds in this video are 25.0, thus the frame in which Julia Gillard's face first appears is frame 142 (5.68 x 25.0).



```
3085830_high_Face0128_Frame000142_X0411_Y0108_col05CD1F.jpeg JPEG 251x251 251x251+0+0 8-bit DirectClass 16.6kb
```

This jpeg file says that the X and Y coordinates are 411 and 108, which is what is stored for that row for the start position. The dimension of that file is 251x251, slightly bigger than what is stored.

The last frame in which Julia Gillard's face is detected for that section is 267 (10.68 x 25.0). The closest stored frame to 267 is 265.

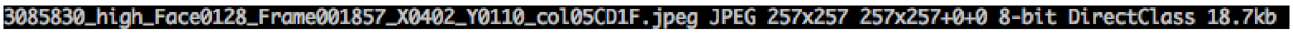


```
3085830_high_Face0128_Frame000265_X0409_Y0121_col05CD1F.jpeg JPEG 231x231 231x231+0+0 8-bit DirectClass 1.5kb
```

This jpeg file says that the X and Y coordinates are 409 and 121, which is not what is stored for that row for the end position. The dimension of that file is 231x231, slightly smaller than what is stored.

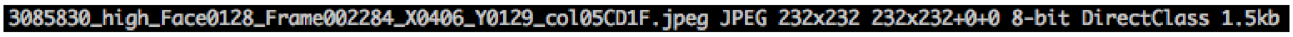
Following the same process for the other two rows the following are the relevant frames:

Row 2 Start Frame: 1857 (74.28 x 25.0)



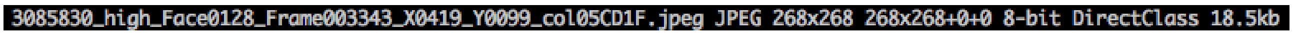
```
3085830_high_Face0128_Frame001857_X0402_Y0110_col05CD1F.jpeg JPEG 257x257 257x257+0+0 8-bit DirectClass 18.7kb
```

Row 2 End Frame: 2284 (91.36 x 25.0)



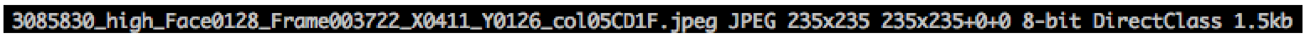
```
3085830_high_Face0128_Frame002284_X0406_Y0129_col05CD1F.jpeg JPEG 232x232 232x232+0+0 8-bit DirectClass 1.5kb
```

Row 3 Start Frame: 3343 (133.72 x 25.0)



```
3085830_high_Face0128_Frame003343_X0419_Y0099_col05CD1F.jpeg JPEG 268x268 268x268+0+0 8-bit DirectClass 18.5kb
```

Row 3 End Frame: 3722 (148.88 x 25.0)



```
3085830_high_Face0128_Frame003722_X0411_Y0126_col05CD1F.jpeg JPEG 235x235 235x235+0+0 8-bit DirectClass 1.5kb
```

RESULTS

	START		END		DIM (START)		DIM (END)	
	X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
STORED	411	108	411	126	244	244	244	244
ACTUAL	411	108	409	121	251	151	231	231

	START		END		DIM (START)		DIM (END)	
	X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
STORED	411	108	411	126	244	244	244	244
ACTUAL	402	110	406	129	257	157	232	232

	START		END		DIM (START)		DIM (END)	
	X	**Y**	**X**	**Y**	**X**	**Y**	**X**	**Y**
STORED	411	108	411	126	244	244	244	244
ACTUAL	419	99	411	126	268	268	235	235

ANALYSIS

After analysing the results which compare the stored values with the relevant values stored in the face tracking file, the bug is clearly evident.

The only values that match are the start position of the first section and the end position of the last section. The dimension values are never found to be correct.

A possible cause could be issues with iterating through the array and storing the wrong values.

CONCLUSION

For the implementation of the HTML face tracking element to output correct information, this bug needs to be fixed.

The start and end position for every record needs to match the correct values. As for the dimension, considering we are only storing one value, I believe the best solution would be to store the largest dimension value for that particular section.