Índice general

Índice de figuras

Capítulo 1

Aplicación web de difusión por YouTube

En este capitulo se describe la primera de las dos aplicaciones que se han diseñado, para cumplir los objetivos.

Esta primera aplicación web, maneja y retransmite contenido captado por una cámara local a eventos de YouTube.La aplicación incorpora las siguientes funcionalidades:

1. Permite crear eventos programados y asociados a un canal de YouTube
2. Permite iniciar y detener la retransmisión del evento a través de ffmpeg
3. Permite añadir subtítulos al evento retransmitido.
4. Muestra en su pantalla principal los eventos en directo asociados a un canal de YouTube.

1.1. Diseño

Antes de pasar a analizar el código con detalle se explica la arquitectura global con el objetivo de entender mejor su funcionamiento.La explicación se apoya en la figura 1.1:

a) Comunicación con YouTube desde NodeJS: para esta comunicación se usan las librerías de Google *YouTube Live Streaming Api* en su versión desarrollada en Pyhton. En esta comunicación se produce el intercambio de datos entre los servidores de YouTube y nuestro servidor.

b) Comunicación con ffmpeg desde NodeJS: Ffmpeg es usado para retransmitir contenido audiovisual hacia los servidores de YouTube. Para el manejo de ffmpeg se ha implementado un script en Python que se encarga de ejecutar ffmpeg a través de la linea de comandos.

c) Obtención del flujo de audiovisual: Como fuente de vídeo y audio se usa una cámara local con un micrófono incorporado.El flujo de vídeo se recoge usando *video4linux*, una API de captura de vídeo integrada en Linux.

d) Comunicación con la interfaz del cliente: La aplicación presenta una interfaz para la interactuación del cliente. Esta interfaz intercambia datos con el nuestro servidor a través de HTTP. Por otro lado para la visualización de vídeo el cliente tiene dos opciones o ir a la pagina web de YouTube donde se muestra el vídeo en emisión o visualizarlo en la propia interfaz de la aplicación web.

e) Comunicación ffmpeg con YouTube: Para que YouTube pueda comenzar su retransmisión, necesita recibir un flujo de vídeo en sus servidores. Ffmpeg es el encargado de enviar este flujo usando el protocolo *RTMP (Real Time Messaging Protocol)*

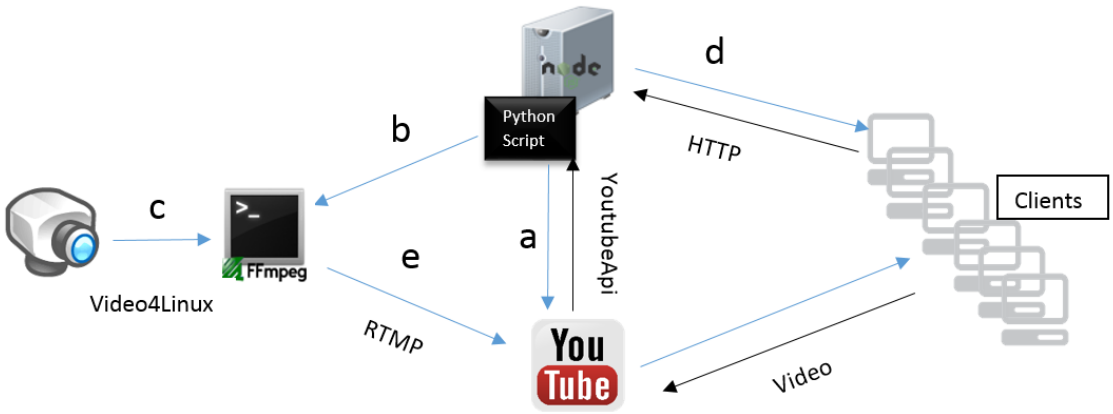

Figura 1.1: Arquitectura SurveillanceApp

1.2. Ejecución de Python en NodeJS

Una de las dificultades encontradas a la hora de desarrollar la aplicación es que las librerías proporcionadas por YouTube no presentan una versión estable para JavaScript. Como se ha comentado anteriormente las funcionalidades relacionadas con YouTube han sido desarrolladas con Python, como se podrá ver más adelante. Para manejar estos scripts se ha recurrido al módulo child_process, que forma parte de NodeJS. A través de este módulo NodeJS nos proporciona la opción de crear un proceso hijo de forma asíncrona, es decir sin bloquear el flujo principal del servidor. Child_proccess proporciona varios métodos para realizar esta tarea, en esta aplicación se usara el método spawn, que es capaz de generar un nuevo proceso a partir de un comando dado, en nuestro caso el comando será el de ejecución de ficheros Python. El método toma tres argumentos **child_process.spawn(command[, args][, options])** , el primero es el comando a ejecutar, el segundo un array con los argumentos del comando si este los tuviera y por ultimo opciones del comando. De esta forma se crea un proceso paralelo al servidor capaz que ejecuta el comando en un terminal del sistema sin bloquear al servidor. A continuación se incluye un fragmento de código donde se puede ver el uso del módulo dentro de la aplicación. En el podemos observar una primera sentencia donde se crea el proceso haciendo uso de spawn , ejecutando el script de Python. Una vez generado el proceso este proporciona dos salidas stdout y stderr.

- **stdout**, recoge los datos devueltos por Python. De esta forma si desde Python queremos enviar datos al flujo principal debemos hacerlo mediante el método print y serán recogidos por stdout.
- **stderr**, aquí se recoge la salida de error tanto para errores de ejecución del comando como errores que puedan surgir dentro del script.

```
exports.retrieveStreamList =  function(req,res,status){

  var process = spawn('python',['./public/python/streamList.py', ' --
     status' , status]);
  processOutput(res,process);
}

function processOutput(res,process){

  var py_data;
  var py_err;

  process.stdout.on("data",function(data){
        py_data += data;
  })

  process.stdout.on("end",function(){

  if(typeof py_data !== "undefined"){
      py_data = py_data.substring(9);
      console.log(py_data)
      if(py_data.localeCompare("ERROR") == 1){
        res.status(500).end();
      }else{
        res.end(py_data)
      }
    }
  })

  process.stderr.on("data", function(data){
      py_err += data;
  })
  process.stderr.on("end",function(){

    if(typeof py_err !== 'undefined'){
      res.status(500).end()
        console.log("Python Error: " + py_err)
    }
  })
}
```

1.3. Servidor NodeJS: Diálogo con YouTube

Para esta comunicación se han usado las librerías proporcionadas por YouTube API. YouTube API proporciona muchas alternativas, para esta aplicación se han elegido dos recursos livebroadcast y livestream ambos explicados en el capítulo tres. Con estos recursos hemos desarrollado tres funcionalidades relacionadas con YouTube en la aplicación, separadas en tres scripts de Python.

1.3.1. Seguridad

```
# The CLIENT_SECRETS_FILE variable specifies the name of a file
    that contains
# the OAuth 2.0 information for this application, including its
    client_id and
# client_secret.

CLIENT_SECRETS_FILE = "./private/client_secret.json"

YOUTUBE_READ_WRITE_SCOPE = "https://www.googleapis.com/auth/
    youtube"
YOUTUBE_API_SERVICE_NAME = "youtube"
YOUTUBE_API_VERSION = "v3"

MISSING_CLIENT_SECRETS_MESSAGE = """
WARNING: Please configure OAuth 2.0

"""
def get_authenticated_service(args):

  flow = flow_from_clientsecrets(CLIENT_SECRETS_FILE,
    scope=YOUTUBE_READ_WRITE_SCOPE,
    message=MISSING_CLIENT_SECRETS_MESSAGE)

  storage = Storage("%s-oauth2.json" % sys.argv[0])
  credentials = storage.get()

  if credentials is None or credentials.invalid:
    credentials = run_flow(flow, storage, args)
```

En este fragmento de código aparecen varias cosas importantes para poder entender el funcionamiento de las librerías. Como se ha comentado en el capítulo tres YouTube ha adoptado Oauth 2.0 como sistema de autenticación. Para poder usar las librerías de YouTube primeramente debes registrarte en la consola de desarrolladores de Google y dar de alta tu proyecto. Una vez dado de alta debes habilitar las API que tu proyecto usará. Terminado este proceso obtendrás el secreto de cliente (clien_secret.json), es un fichero el cual contiene las credenciales que te identifican como propietario de la cuenta de Google [1] asociada al proyecto.

[1] https://console.developers.google.com

Otra parte importante es la de los ámbitos o *scope*. Esta parte especifica qué permisos tiene para interactuar con YouTube la aplicación. Por ejemplo, en el código escrito arriba el *scope* permite tanto leer como modificar los recursos asociados a la cuenta de YouTube. Los otros parámetros indican el API que va a ser usada y su versión.

A continuación nos encontramos con el método get_authenticated_service, que lleva a cabo el proceso de autenticación. Dicho método es proporcionado por Google en la documentación de la API. Este método busca un fichero con las credenciales que autorizan a la aplicación a acceder a los datos almacenados en la cuenta de YouTube asociada a la misma. Si es la primera vez que se accede estos datos no estarán generados por lo cual se redirigirá automáticamente al usuario a los servidores de autenticación de Google. Si la autenticación ha sido correcta se creará automáticamente un fichero que permite el acceso a la cuenta por parte de la aplicación.

1.3.2. Creación de eventos

Una de las funcionalidades que nos permite la aplicación es la creación de eventos programados. Para ellos a través de un formulario contenido en la interfaz web son recogidos una serie de datos como el título del evento, el horario, la privacidad o la calidad del mismo, todas ellas propiedades de ambos recursos. Estos datos son enviados al servidor en formato JSON y proporcionados como entrada a createEvent.py. CreateEvent hace uso del método insert tanto de livebroadcast como de livestream para crear ambos recursos. Una vez creados se recupera su ID para proporcionárselo al método bind perteneciente a livebroadcast de forma que el evento es creado en el canal asociado.

```
    # Create a liveBroadcast resource and set configuration
    def insert_broadcast(youtube, options):
      insert_broadcast_response = youtube.liveBroadcasts().insert(
       part="snippet,status,contentDetails",
        body=dict(snippet=dict(
            title= args["broadcast_title"],
            scheduledStartTime= args["start_time"]),

          status=dict(
            privacyStatus= args["privacy"]),
          contentDetails=dict(
            monitorStream=dict(
              enableMonitorStream = 'true')
            )
    )
      ).execute()

    snippet = insert_broadcast_response["snippet"]
    return insert_broadcast_response["id"]

    # Create a liveStream resource and set configuration
    def insert_stream(youtube, options):
        insert_stream_response = youtube.liveStreams().insert(
            part="snippet,cdn",
            body=dict(
```

```
                snippet=dict(
                    title= args["broadcast_title"]
                ),
                cdn=dict(
                    format= args["format"],
                    ingestionType="rtmp"
                )
            )
        ).execute()

    snippet = insert_stream_response["snippet"]
    return insert_stream_response["id"]

    # Bind the broadcast to the video stream.
    def bind_broadcast(youtube, broadcast_id, stream_id):
        bind_broadcast_response = youtube.liveBroadcasts().bind(
        part="id,contentDetails",
        id=broadcast_id,
        streamId=stream_id
        ).execute()
```

1.3.3. Recuperación de información de evento

Dentro de la aplicación en su pantalla principal podemos ver el vídeo de aquellos eventos que se encuentran en directo sin necesidad de acudir a la web de YouTube directamente, esto es posible gracias a que YouTube mediante su propiedad monitorStream nos proporciona un i-frame el cual tiene como fuente(source) un reproductor de YouTube embebido asociado al nuestro evento.

Por otro lado dentro de la parte privada de la aplicación, únicamente accesible para administradores, se pueden ver los eventos programados e iniciar su retransmisión o los eventos que se están retransmitiendo para detenerlos.

Todas estas funcionalidades requieren de datos proporcionados por los servidores de YouTube. Para recuperar estos datos se recurre al método list, perteneciente tanto al recurso livebroadcast como al livestream. Dicho método admite filtros para buscar los eventos que nos interesan. En nuestro caso filtramos por el estado del evento, activo o programado, y además especificamos las partes del recurso que queremos recuperar por último se limitan los resultados a 50.

```
    list_broadcasts_request = youtube.liveBroadcasts().list(
        broadcastStatus= status,
        part="snippet, contentDetails",
        maxResults=50
          )
```

En el caso de livestream recuperamos únicamente el flujo asociado a la retransmisión mediante su ID, de este recurso nos interesa su propiedad *cdn* ya que de ahí obtenemos el *stream_name*, necesario para la posterior retransmisión ya que este ID identifica al recurso [2] en los servidores de ingestión de contenido de YouTube, además

[2] Capítulo 3. Propiedades livestream y livebroadcast

recuperamos los datos de ingestión como la calidad del evento de forma que luego se pueden calcular datos como bitrate o resolución. Otras propiedades importantes que recuperamos son *monitor* que contiene un *iframe* que posteriormente se inserta en la aplicación web para poder visualizar el evento una vez activo. Una vez recuperados son almacenados en list_broadcast_request. Este objeto contiene la información de todos los eventos y de él recuperamos la información que nos interesa de cada uno y formamos un JSON que será la respuesta a la petición web.

```
    while list_broadcasts_request:
        broadcast_response = list_broadcasts_request.execute()
        for broadcast in broadcast_response.get("items", []):

        title = broadcast["snippet"]["title"]
        monitorStream = broadcast["contentDetails"]["monitorStream"][
            "embedHtml"]
        stream_id = broadcast["contentDetails"]["boundStreamId"]
        cdn = getStreamKey(youtube,stream_id)
        data_output["data"].append({"title" : title , "streamkey" :
            cdn["ingestionInfo"]["streamName"],
        "monitor": monitorStream, "quality" : cdn["format"],"
            broadcastID": broadcast["id"] })

        list_broadcasts_request = youtube.liveBroadcasts().list_next(
            list_broadcasts_request, broadcast_response)

    # Retrieve a livestream resource match with stream_id
    def getStreamKey(youtube,stream_id):

        list_streams_request = youtube.liveStreams().list(
        part="cdn",
        id=stream_id,
        maxResults=1
        )
        list_streams_response = list_streams_request.execute()
        return list_streams_response["items"][0]["cdn"]
```

1.3.4. Terminar la retransmisión del evento

Para conseguir este objetivo se ha usado el método transition perteneciente al recurso livebroadcast. Con dicho método recuperamos una emisión en directo a partir de su ID y modificamos su estado de activo a finalizado, de esta forma YouTube dará por terminada la retransmisión y el evento finalizará. También debemos acabar con la ejecución de ffmpeg ya que mientras este se este ejecutando no podremos iniciar la retransmisión de un nuevo evento ya que la cámara estará ocupada por ffmpeg , para ello usamos el comando -pkill ffmpeg, el cual acabará con el proceso.

```
    def stop_broadcast(youtube,brID):
```

```
    broadcast_request = youtube.liveBroadcasts().transition(
      broadcastStatus= "complete",
      id = brID,
      part = "status")
    broadcast_request.execute()
```

1.4. Servidor NodeJS: Comunicación con Ffmpeg

Ffmpeg es el encargado de enviar el flujo audiovisual a los servidores de ingestión de contenido de YouTube. Ffmpeg presenta varias herramientas pero para este objetivo únicamente hemos usado la funcionalidad que permite manejarlo desde la terminal. Para ello se ha desarrollado un script de Python que escribe en un fichero .sh el comando de ffmpeg a ejecutar, tras esto mediante la instrucción os.system se ejecuta el fichero. Para la construcción del comando ffmpeg se toma como entrada primeramente la calidad del evento, en función de esta calidad se añade el *bitrate* y la resolución del vídeo según las especificaciones de YouTube. Otro parámetro fundamental que recibe el script es el stream_key, representa un identificador único del *stream* dentro de los servidores de YouTube para que éste pueda asociar el flujo audiovisual recibido al evento.

```
def list_streams(stream_key,resolution,bitrate):

os.system("chmod +rw ./public/static/ffmpeg.sh")
outfile = open('./public/static/ffmpeg.sh', 'w')
outfile.write('ffmpeg -f alsa -ac 2 -i default -f video4linux2
    +    -framerate 15 -video_size ' + resolution +
'-i dev/video0 -vcodec libx264 -preset veryfast -minrate ' +
    bitrate  + '-maxrate 1000k -bufsize 1000k' +' -vf "format=
    yuv420p"  -g  30 ' + '-vf drawtext= "fontfile=/usr/share/fonts
    /truetype/freefont/FreeSerif.ttf:   +   fontsize  =24:''
    fontcolor=yellow:textfile=./public/static/subtitles.txt:reload
    =1 :x=100:y=50"' +
' -c:a libmp3lame -b:a 128k -ar 44100' +
' -force_key_frames 0:00:04   f   flv rtmp://a.rtmp.youtube.com/
    live2/' + Stream_key
outfile.close()
os.system("chmod 0755 ./public/static/ffmpeg.sh")
os.system("./public/static/ffmpeg.sh")
```

Ffmpeg presenta una gran variedad de posibilidades a continuación se explican las principales partes del comando usado en el script:

- **Video4Linux y ALSA**, son componentes pertenecientes al SO Linux que se encargan de recoger los flujos de vídeo y audio respectivamente. Como fuente de vídeo usamos la cámara principal del sistema, dev/video0/, para el audio el micro definido por defecto.
- **Libx264**, *codec* de vídeo H264, tras él aparecen las opciones de codificación del

vídeo, el perfil no está especificado pero por defecto ffmepg usa el perfil básico del *codec*.

- **Drawtext**, opción usada para superponer texto sobre el vídeo, es la equivalencia a los subtítulos. El texto es leído de un fichero .txt, que tiene asociada la opción *reload* encargada de detectar el cambio en el fichero de texto. Las otras opciones lo acompañan se refieren al formato del texto mostrado en pantalla.
- **Force_key_frames**, a la hora de codificar el vídeo se usan *frames* de referencia a partir de los cuales se codifican los siguientes, este comando obliga a enviar estos fotogramas/marcos cada cuatro segundos.
- **Rtmp**, protocolo de comunicación usado entre el servidor de YouTube y ffmpeg para el intercambio del contenido.

1.5. Dialogo HTTP e interfaz de cliente

La aplicación cuenta con un interfaz para interactuar con canales de YouTube, pudiendo realizar todas las operaciones descritas anteriormente.

diagrama

1.5.1. Seguridad

La aplicación tiene acceso a datos privados y acciones de la cuenta de YouTube del usuario por lo que es necesario implementar mecanismos de seguridad para que solo el usuario tenga acceso a ellos.Por ello la aplicación tiene una parte pública y otra privada solo accesible por el administrador.

Como mecanismo de seguridad se ha usado un middleware, implementado en el servidor. El *middleware*, intercepta las peticiones HTTP que dan acceso a la parte privada, donde se encuentran las operaciones de crear,iniciar o detener eventos.

```
//rutas donde se compureba si el usuario tiene permiso
server.use("/createbroadcast",security.middleware)
server.use("/streams",security.middleware)
```

El *middleware* comprueba si el usuario ha iniciado una sesión previamente, en cuyo caso se permitirá el acceso a la dirección requerida. En caso contrario sera redirigido a un formulario de *login*, donde podrá iniciar sesión.

Por último una vez iniciada la sesión se da la opción al usuario de cerrarla.

```
var sess;

exports.middleware = function(req,res,next){
  sess = req.session
  if(sess.auth){
      next(); //if log ok ==> continue
  } else{
```

```
      console.log("Incorrect User")
      res.redirect("/login")

  }
}

exports.login = function(req,res,session,config){

  if(req.body.user == config.admin.name && req.body.password ==
      config.admin.password){
      sess = req.session;
      console.log("Right User : session started")
      sess.auth = "true";
      res.end("Right user")
  } else{
    res.status(400).end("User or Password Error");
  }
}

exports.logout = function(req,res){
  req.session.destroy();
  res.redirect("/login")
}
```

1.5.2. Diálogo HTTP

La comunicación con el servidor NodeJS se lleva a cabo a través del protocolo HTTP. Esta comunicación ha sido desarrollada con AngularJS, *framework* del lenguaje JavaScript. Principalmente la comunicación HTTP se encarga de transmitir los datos introducidos por el usuario en el interfaz al servidor, que posteriormente se los enviara a YouTube.

Angular usa los llamados controladores para manejar los datos de la aplicación, por ello se ha desarrollado un controlador por cada pantalla que se encarga de recoger y enviar los datos al servidor y mostrar su respuesta en caso de que fuera necesario.

A continuación se muestran dos controladores el primero de ellos es el encargado de mostrar las retransmisiones en directo, el segundo muestra la lista de eventos programados y que pueden empezar su retransmisión.

```
var app = angular.module('app', []);

  //index.html controller
  app.controller("indexController",['$scope', '$http','$sce',function
      ($scope,$http,$sce){
    $scope.setNavbar = "static/navbar.html"

    $scope.videos = {};
    //Add iframe element to index.html
    $scope.addVideo = function(video){
      if(video.length > 0){
        //Extract iframe src
```

```
        src = video.substring(video.indexOf('embed') + 6 ,video.
            indexOf('?'))
        return $sce.trustAsResourceUrl("https://www.youtube.com/embed
            /" + src)
      }
    }

    $scope.getVideoList = function(){
      $http({
        method:'GET',
        url : '/videolist'
      }).success(function(response){
        if(response.data.length == 0){
           $scope.msg = "No avalaibles events now"
        }else{
          $scope.videos = response.data;
        }

      }).error(function (response){
        $scope.msg = "Can t show the videos"
      })
    }
    $scope.getVideoList();
  }])
```

```
    //stream_list.html controller
  app.controller("dataListController",['$scope','$http','$window',
      function($scope,$http,$window){
    $scope.events = {}
    $scope.getList = function(){
      $http({
        method: "GET",
        url: "/streamlist"
      }).success(function(response){
        $scope.events = response;
     }).error(function(response){
        $scope.msg = "Can't show video list"
     })
    }

    $scope.startStream = function(streamkey,quality){
      dataSend = {"streamkey" : streamkey,"quality": quality}
      $http({
        method: "POST",
        url: "/startstreaming",
        data: dataSend,
        headers :  {'Content-Type': 'application/JSON'}
      }).success(function (response){
        console.log(response)
        $window.location.href = '/subtitles';
      }).error(function(response){
        $scope.msg = "Can't stop the event"
      })
```

```
    }
    $scope.getList();
  }])
```

1.5.3. Interfaz

La aplicación implementa una sencilla interfaz gráfica desde la cual se puede interactuar con YouTube. Esta interfaz que consta de dos partes, una publica y otra privada.

Usa angular para introducir o recibir los datos en las comunicaciones HTTP gracias al *scope*,un contenedor que almacena el modelo de datos del controlador.

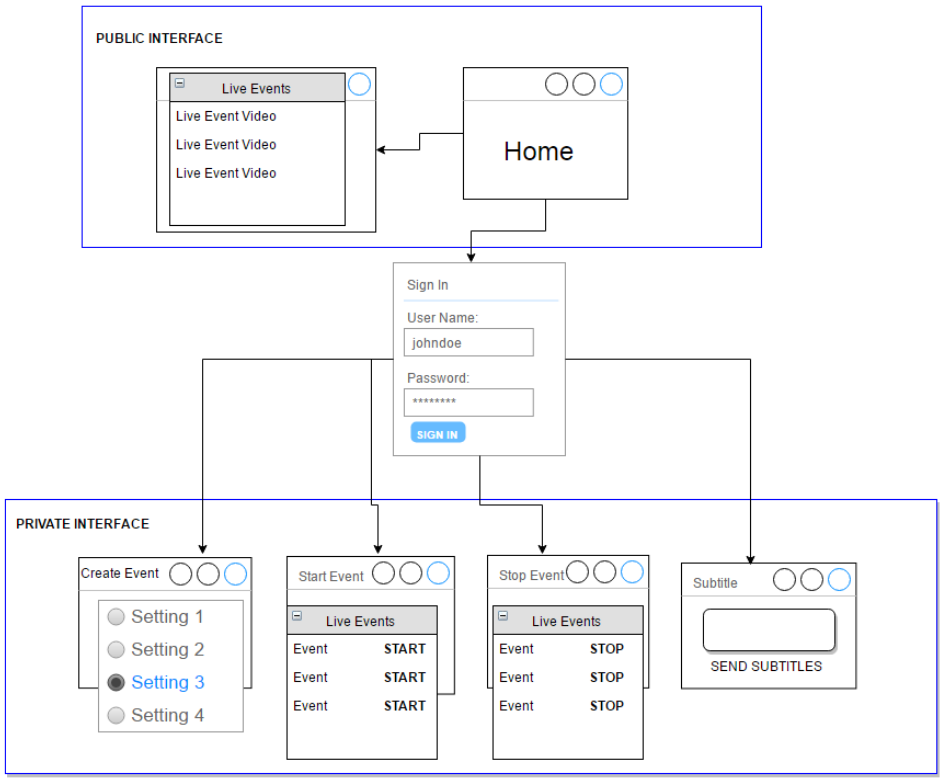

Figura 1.2: Diagrama de la Aplicación

La primera parte es una parte pública, donde se pueden reproducir los vídeos de los eventos en directo en ese momento, para conseguir esto se ha usado el elemento de vídeo embebido.

```
<!doctype html>
<html ng-app="app">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs
       /1.3.3/angular.min.js"></script>
```

```
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.19/
       angular-cookies.js"></script>
    <link rel="stylesheet" type="text/css" href="static/css/
       indexStyle.css">
    <script src="app.js"></script>
  </head>
  <body ng-controller = "indexController">
      <div ng-include = setNavbar ></div>
      <img src="/static/images/spy.png"/>
      <div  class="container">
        <table >
          <tr>
            <th>Title</th>
            <th> Video</th>
          </tr>
          <tr ng-repeat="video in videos">
            <td>{{video.title}}</td>
            <td><iframe ng-src="{{addVideo(video.monitor)}}"></iframe
               ></td>
          </tr>
        </table>
        <p class="error">{{msg}}</p>
      </div>
  </body>
</html>
```

La segunda parte es privada, esta protegida por el middleware comentado anteriormente. Una vez superada la seguridad, desde el interfaz se pueden crear eventos seleccionando la hora de inicio y calidad del evento, se muestra también una lista de todos los eventos programados, junto con la opción de empezar estos eventos. Si dicha opción es seleccionada ffmpeg comenzara la retransmisión de vídeo, y desde el interfaz se ofrecerá la posibilidad de insertar subtítulos en la emisión. Para ello el contenido enviado al servidor es escrito en un fichero de texto que sera leído por ffmpeg. La última funcionalidad implementada es la de dar por finalizado el evento, de forma que aparecerá una lista con los eventos en directo del canal en ese momento y solo se deberá seleccionar aquel que queramos finalizar.

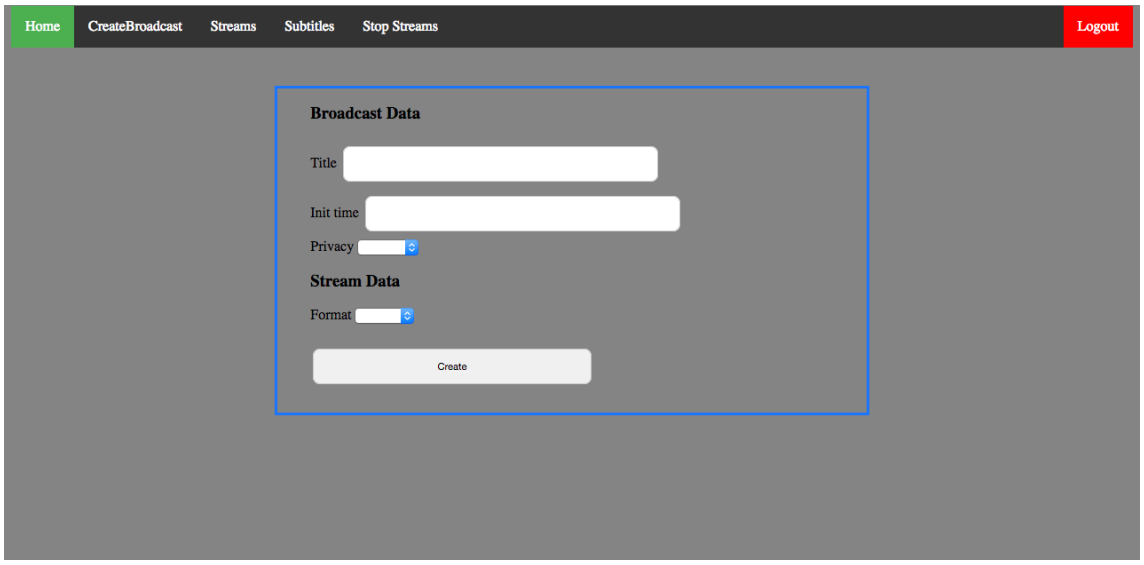

Figura 1.3: Interfaz Creación de Eventos

1.6. Experimentos

Para probar el correcto funcionamiento de la aplicación se ha levantado el servidor NodeJS una IP pública. El equipo tenia conectada una cámara web desde la cual se captaban el flujo de vídeo enviado a YouTube.

Desde un ordenador remoto se ha accedido a la dirección en la que estaba ejecutándose el servidor. Una vez conectado se ha creado, iniciado y finalizado eventos de YouTube satisfactoriamente a través de la aplicación web.

Con estos resultados se puede decir que la aplicación funciona correctamente[3].

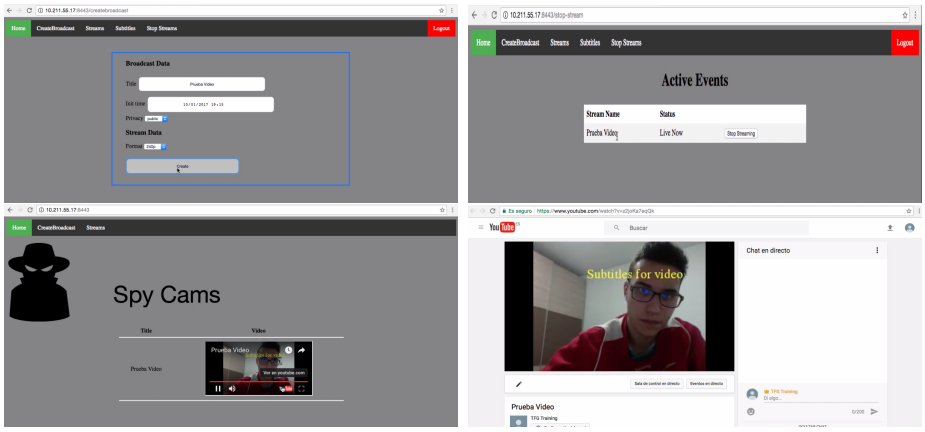

Figura 1.4: Experimento

[3]http://jderobot.org/Apavo-tfgSurveillance_App