



# Trabajo de Fin de Grado

## Sistema de Video Vigilancia remoto, controlable y seguro

*“Remote, controlable and secure  
Video surveillance system”*

*Noel Padrón Díaz*

La Laguna, 3 de septiembre de 2018

**D. Cándido Caballero Gil**, con N.I.F. 42.201.070-A profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

**D. Jonay Suárez Armas**, con N.I.F. 78.639.262-P profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como cotutor

## **C E R T I F I C A ( N )**

Que la presente memoria titulada:

*“Título del Trabajo”*

ha sido realizada bajo su dirección por **D. Noel Padrón Díaz**,  
con N.I.F. 78.645.284-G.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 3 de septiembre de 2018

## Agradecimientos

XXX

XXX

XXX

XXX

# Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial 4.0 Internacional.

## Resumen

El objetivo de este trabajo ha sido la creación de un sistema de video vigilancia, pensado tanto para entornos industriales como domésticos, mediante el cual de forma remota podremos comprobar si hay algún movimiento donde tengamos instalado el sistema, y poder mover la cámara en cualquier dirección a través de un servidor.

Este proyecto se divide en tres grandes partes, una primera de creación de una aplicación para móvil, que detecte movimiento y que se comuniquen con un servidor. Una segunda que consiste en configurar este servidor remoto, su front-end y las conexiones que se realicen a él. Y una tercera parte que incluye un robot aspirador “Roomba”, que servirá de soporte a dicho sistema.

El hardware con el que contaremos será un móvil con una versión de Android que soporte la aplicación programada y un robot aspiradora “Roomba”.

Los lenguajes de programación usados han sido Android, con el IDE “Android Studio”, HTML, usado con el editor de textos “Atom”, y JavaScript junto con Node.js.

**Palabras clave:** Android, HTML, video vigilancia, control remoto, Roomba, seguridad, servidor remoto.

# Abstract

The objective of this project has been the creation of a video surveillance system, designed for both industrial and domestic environments, through which remotely we can check if there is any movement where we have installed the system, and can able to move the camera in any direction through a server.

This project is divided into three parts, a first of creating an application for mobile, which detects movement and communicates with a server. A second one is to configure this remote server, the front-end and the connections that we made to it. And a third part that includes a robot vacuum cleaner "Roomba", which will support this system.

The hardware with which we will be a mobile with an Android version that supports the scheduled application and a vacuum robot "Roomba".

The programming languages used have been Android, with the IDE "Android Studio", HTML, used with the text editor "Atom", and JavaScript along with Node.js.

**Keywords:** Android, HTML, video surveillance, remote control, Roomba, security, remote server.

# Índice general

<b>Capítulo 1 Introducción .....</b>	<b>1</b>
1.1 Contexto .....	1
1.2 Antecedentes.....	1
1.3 Motivaciones .....	2
<b>Capítulo 2 Fases del proyecto.....</b>	<b>3</b>
2.1 Estudio de las herramientas.....	3
2.2 Desarrollo de la detección de movimiento.....	3
2.3 Desarrollo del Streaming.....	3
2.4 Desarro de servidor web.....	3
2.5 Integración con firebase .....	3
<b>Capítulo 3 Herramientas utilizadas.....</b>	<b>4</b>
3.1 Wowza Streaming.....	4
3.2 Android Studio.....	5
3.3 Node.js.....	6
3.4 Jade .....	7
3.5 Firebase.....	8
3.6 Roomba.....	9
3.7 Atom .....	10
3.8 Github .....	11
<b>Capítulo 4 Desarrollo del proyecto .....</b>	<b>12</b>
4.1 Streaming.....	12
4.2 Acople de la detección de movimiento .....	14
4.3 Comunicación con el servidor web .....	16
4.4 Integración con Firebase .....	18
4.4 Roomba.....	20

<b>Capítulo 5 Problemas encontrados .....</b>	<b>6</b>
<b>Capítulo 6 Conclusiones y resultados .....</b>	<b>7</b>
<b>Capítulo 7 Sumary and Conclusion.....</b>	<b>8</b>
<b>Capítulo 8 Presupuesto .....</b>	<b>9</b>
8.1 Estimación coste de herramientas .....	9
8.2 Estimación coste de desarrollo e implantación de la aplicación.....	9
8.3 Presupuesto final estimado.....	9
<b>Capítulo 9 Bibliografía.....</b>	<b>11</b>



# Índice de figuras

Figura 1.1: Ejemplo ..... 2

Figura 2.1: Otra figura..... 3

# Índice de tablas

Tabla 1.1: Estimación coste de herramientas .....	2
Tabla 1.2: Estimación coste de desarrollo e implantación de la aplicación.....	2

# Capítulo 1

## Introducción

### 1.1 Contexto

La seguridad es un tema crucial para cualquier empresa o usuario particular para un uso doméstico. Hoy en día, en tema de informática se destaca mucho la ciberseguridad, pero no se puede/debe dejar de lado la seguridad física. Y esta seguridad no está reñida con la informática, no tiene que quedarse todo anclado en el clásico sistema de multicámara para poder tener una visión del espacio que queremos proteger. No es necesaria tanta inversión, ni tanto montaje si podemos conseguir acoplar dos tecnologías que aun sus objetivos parecen bastante distantes, nos permiten otro común aunando fuerzas entre ellos.

### 1.2 Antecedentes

La video vigilancia es una rama que ha estado presente en el día a día de muchas empresas e incluso a nivel doméstico desde que se empezaran a usar cámaras, en blanco y negro, allá por los años 40. Se suelen usar detectores de movimiento en estos sistemas de video vigilancia, en el siglo pasado se usaba el efecto doppler, pero hoy en día es más sencillo, se comparan dos fotogramas en un cierto periodo de tiempo, y si pasa un cierto umbral de movimiento salta la alarma.

En cuanto a los robots Roomba, estos salieron al mercado en el 2002, incluyendo año a año detalles como más sensores de movimiento y acumuladores de residuos, aunque esta última parte no nos Grado en Ingeniería Informática interesa mucho. Con el tiempo han ido evolucionando, y hoy en día se les puede añadir un módulo para controlarlos remotamente, ya sea por wifi o por bluetooth.

En cuanto al streaming, no se realizó la primera conexión hasta mediados de los 90 gracias a programas como RealPlayer, y en este sentido se ha avanzado mucho, ya que en ese entonces se consumía demasiado ancho de banda, y hoy en día con las nuevas tecnologías de compresión de video y de transmisión de datos se puede hacer streaming hasta con un ancho de banda 3G.

Además este proyecto parte de una base de un proyecto anterior, con el propósito de aplicar el sistema robótico desarrollado a los nuevos robots Roomba, conservando el detector de movimiento de la cámara fija.

## **1.3 Motivaciones**

La idea de juntar dos tecnologías, con objetivos finales tan distantes, como la grabación o control de video y la limpieza. Esto supone un reto y una solución diferente a lo que actualmente se ve.

# Capítulo 2

## Fases del proyecto

### 2.1 Estudio de las herramientas

El primer paso fue la toma de contacto con algunas de las herramientas que iba a usar, como **Android Studio**, un IDE del entorno de desarrollo Android, **Wowza**, una herramienta que nos permite hacer streamings desde un servidor local, **Atom**, un editor de textos o Github para tener un control de versiones del mismo.

### 2.2 Desarrollo de la detección de movimiento

Donde se implementará un sistema de detección de movimiento para que estando el móvil efectuando el streaming y detecte un cambio entre dos frames envíe una alerta al servidor donde estará ejecutándose la aplicación web, mediante el servicio de Firebase.

### 2.3 Desarrollo del streaming

Usando la herramienta Wowza, empezaremos a integrar el streaming con el proyecto en Android Studio donde tenemos desarrollado la detección de movimiento. También configurar el servidor local de la herramienta, creando un canal para el streaming y configurando las direcciones y los puertos.

### 2.4 Desarrollo del servidor web

Usando el editor de texto Atom y el entorno de ejecución para javascript Nodejs, usando un socket para iniciar la comunicación entre la aplicación y el servidor. Para la interfaz web se usara el lenguaje de plantillas JADE.

### 2.5 Integración con Firebase

Prescindir del socket de comunicación entre la web y la app usando Firebase, lo que nos garantizará la seguridad que la herramienta de google nos ofrece, aparte de su respuesta en tiempo real.

# Capítulo 3

## Tecnologías y herramientas utilizadas

### 3.1 Wowza Streaming Engine

Es un software de servidor que se utiliza para la transmisión de vídeo en vivo y bajo demanda, audio y aplicaciones RIA (Rich Internet Applications) a través de redes IP públicas y privadas en el escritorio, portátiles y Tablet PC, dispositivos móviles, IPTV set-top boxes conectados a Internet televisores y otros dispositivos conectados a la red. Transmisión en vivo de vídeo y audio que utiliza RTMP, RTP / RTSP y codificadores MPEG-TS.

El servidor se aloja en la red local, permitiendo acceder al panel de control desde una dirección local.

**Wowza Streaming Engine MANAGER** Home Server Applications noel Help Sign Out

## Welcome to Wowza Streaming Engine!

Trial License (Expires sep 3, 2018) - 1 day left [See License Details](#)

### Status

**Connections** Incoming and outgoing

**Usage** CPU, Memory, Heap and Disk

**Server Uptime**  
Since 30 Aug 2018 11:29:04 AM

**Features**  
Transcoder: **Licensed**  
DRM: **Licensed**  
nDVR: **Licensed**

**Test Video**

To play an on demand test video, click **Test Players**.

[▶ Test Players...](#)

**Performance Warning!** You are currently using *Developer* performance settings. If this server is running in a production environment, switch to *Production* performance settings on the [Java Settings](#) page.

**4.7.6 Update Available**  
You are using **version 4.7.5**. A new update for Wowza Streaming Engine is available.  
[More Info and Download](#)

**Ready to Purchase?**  
[Buy Now](#)

**Application Connection Settings**  
Use the following settings to publish a stream to the Wowza Streaming Engine server:

<b>Host - Server</b>	169.254.169.189
<b>Host - Port</b>	1935
<b>Application</b>	A live application name on this server
<b>Stream Name</b>	The stream name you want to use
<b>Login</b>	A valid source user name and password

**Getting Started With Applications**  
Wowza Streaming Engine software uses **applications** to deliver streaming content. An application is a set of settings for live or video on demand (VOD) streaming. Either use the preinstalled default applications or go to the [Add Application](#) page to easily create and configure new applications.

**Live Applications**  
A [live](#) streaming application is preinstalled to allow you to easily publish video

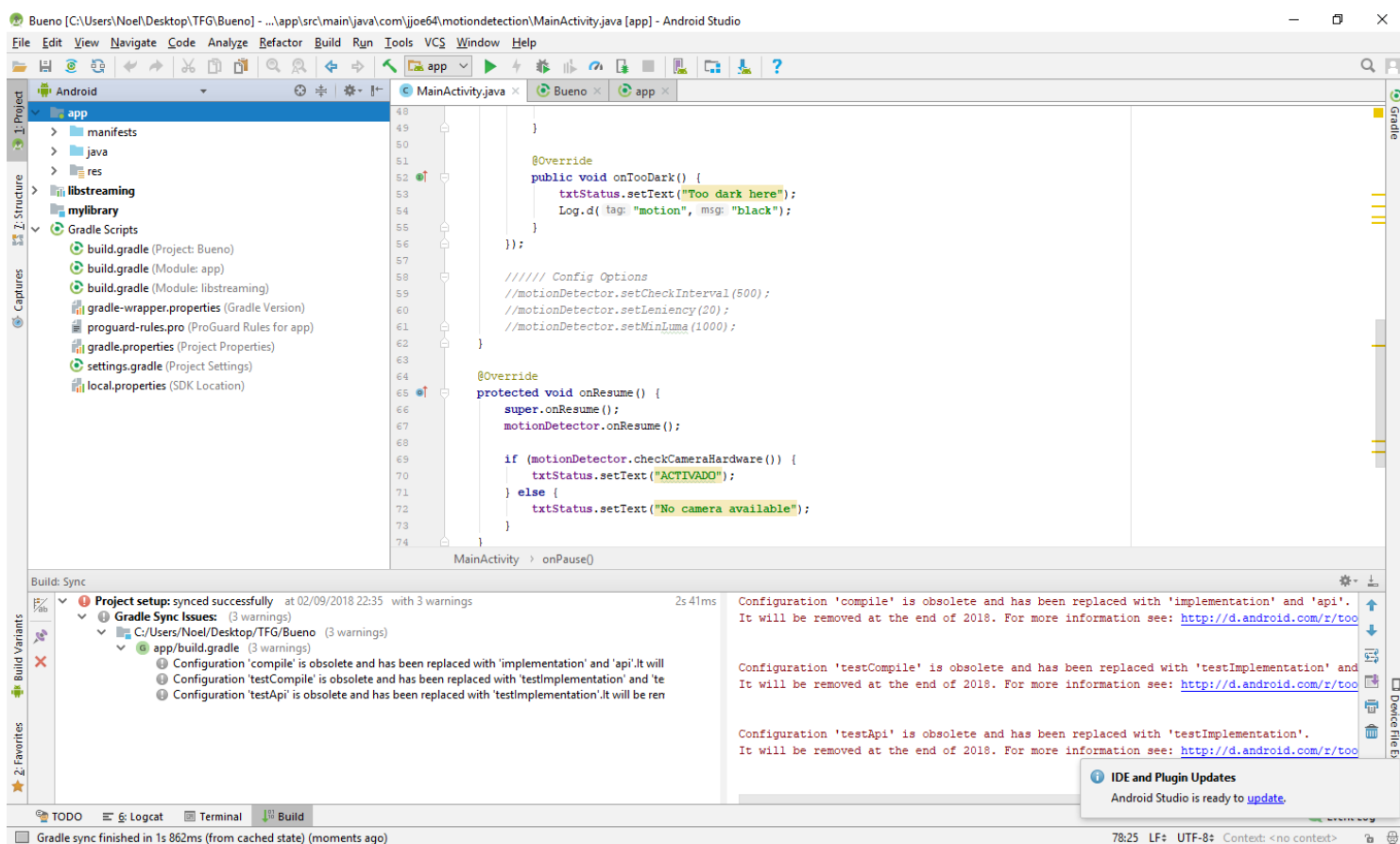
## 3.2 Android Studio

Es el IDE oficial de Android para el desarrollo de aplicaciones. Basado en IntelliJ IDEA; un entorno o ambiente de desarrollo para programas, que posee potentes herramientas de edición de código. Se puede decir que en cuanto a su análisis de código el mismo destaca los errores de forma inmediata, para así dar una solución más rápida de estos. Como herramientas integradas para el desarrollo o construcción de programas en Android, contiene una interfaz de usuario que es construida o diseñada previamente, con variados modelos de pantalla, donde en ella los elementos existentes pueden ser desplazados.

Android Studio posee distintos componentes que ayudan a la tarea de la construcción de aplicaciones; sistema de construcción basado en Gradle, la construcción de variantes y múltiples archivo APK, como también plantillas de código que ayudan a la creación de aplicaciones.

En cuanto al desarrollo del flujo de trabajo, Android Studio posee un conjunto de herramientas encargadas, Adicionando a eso el posible acceso desde la línea de comandos las herramientas SDK.

Lomas importante de todo esto es que, Android Studio ofrece comodidad para el desarrollo, ya que desde él es posible invocar, durante el desarrollo de aplicaciones, las herramientas necesarias como una forma más ágil de trabajo.



### 3.3 Node.js

Es un entorno que trabaja en tiempo de ejecución, de código abierto, multi-plataforma, que permite a los desarrolladores crear toda clase de herramientas de lado servidor y aplicaciones en JavaScript. La ejecución en tiempo real está pensada para usarse fuera del contexto de un explorador web (es decir, ejecutarse directamente en una computadora o sistema operativo de servidor). Como tal, el entorno omite las APIs de JavaScript específicas del explorador web y añade soporte para APIs de sistema operativo más tradicionales que incluyen HTTP y bibliotecas de sistemas de ficheros.

Node fue lanzado inicialmente, sólo para Linux, en 2009. El gestor de paquetes NPM fue lanzado en 2010, y el soporte nativo para Windows fue añadido en 2012. La versión actual LTS (Long Term Support) es Node v8.9.3 mientras que la última versión es Node 10.

Node.js posee métodos para especificar que función ha de ser llamada dependiendo del verbo HTTP usado en la petición (GET, POST, SET, etc.) y la estructura de la URL ("ruta"). También tiene los métodos para especificar que plantilla ("view") o gestor de visualización utilizar, donde están guardadas las plantillas de HTML que han de usarse y como generar la visualización adecuada para cada caso.

También permite crear módulos, APIs asíncronas y manejadores de rutas entre muchas otras cosas.





## 3.4 Jade

Es un lenguaje de plantillas, un binario de node.js, y una forma rápida de generar contenido. Básicamente define una estructura semántica, ordenada y jerárquica, usando solamente el nombre de las etiquetas HTML, de esta forma solamente nos dedicamos a generar contenido y no en aprender un lenguaje nuevo, dado que es una abstracción de HTML plasmada en un formato más sencillo, de aquí deriva el “nuevo” lenguaje llamado Jade.

Básicamente el código se reduce considerablemente debido a que se escribe una sola vez la etiqueta deseada, evitando así abrir y cerrar etiquetas HTML. La indentación de dos espacios define la jerarquía, y como resultado natural se obtiene una plantilla definida con orden.

```
!!! 5
html(lang="en")
  head
    title= pageTitle
    :javascript
      | if (foo) {
      |   bar()
      | }
  body
    h1 Jade - node template engine
    #container
      - if (youAreUsingJade)
        You are amazing
      - else
        Get on it!
        Get on it!
        Get on it!
        Get on it!
```

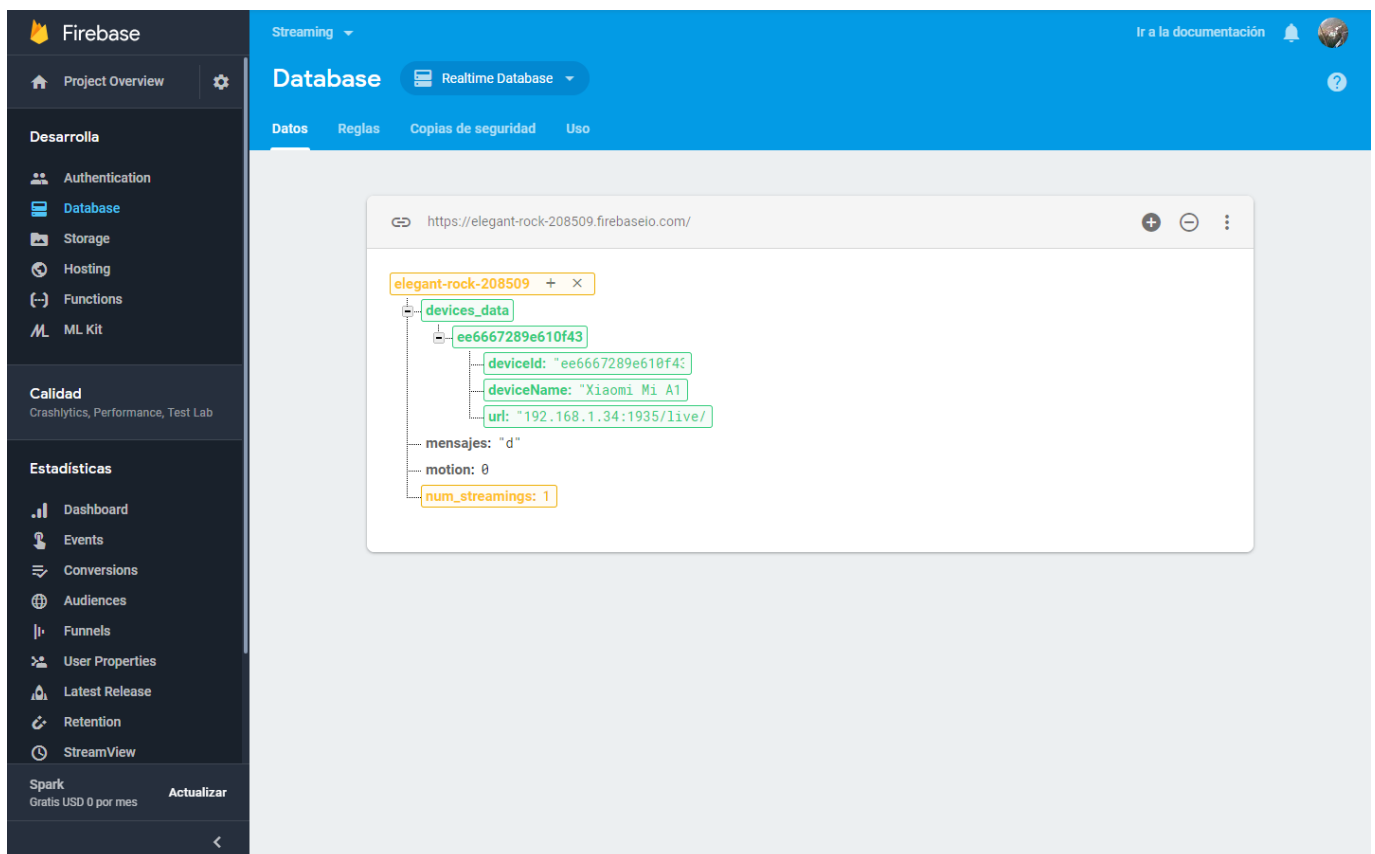
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      //
      if (foo) {
        bar()
      }
      //]]&gt;
    &lt;/script&gt;
  &lt;/head&gt;
  &lt;body&gt;
    &lt;h1&gt;Jade - node template engine&lt;/h1&gt;
    &lt;div id="container"&gt;
      &lt;p&gt;You are amazing&lt;/p&gt;
    &lt;/div&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="489 909 506 926" data-label="Page-Footer"><p>7</p></div>
```

## 3.5 Firebase

Firebase es una plataforma de back-end para construir aplicaciones móviles y web, se encarga del manejo de la infraestructura permitiendo que el desarrollador se enfoque en otros aspectos de la aplicación. Entre sus características se incluye base de datos de tiempo real, autenticación de usuarios y almacenamiento (hosting) estático. La idea de usar Firebase es no tener que escribir código del lado del servidor y aprovechar al máximo las características que nos provee la plataforma.

En esta aplicación lo que más interesa es la base de datos en tiempo real, para proveer la dirección del streaming, el control del Roomba y la detección de movimiento.

Google permite el acceso al panel de control siendo usuario de una cuenta Gmail, pudiendo acceder a la base de datos y a las demás funciones.



## 3.6 Roomba

Es un aspirador robótico fabricado y vendido por iRobot. El Roomba se lanzó al mercado en 2002, y se calcula que en febrero de 2014 ya se habían vendido más de 10 millones de unidades en todo el mundo. El robot aspirador Roomba incluye una serie de sensores (táctiles, ópticos y acústicos, dependiendo de cada serie y modelo) que le permite, entre otras cosas, detectar obstáculos, acumulaciones de residuos en el suelo y desniveles pronunciados tales como escaleras. Utiliza dos ruedas motrices independientes que le permiten ejecutar giros de 360 grados. Adicionalmente, se le puede programar para realizar otras funciones más “creativas” mediante un ordenador y haciendo uso de la denominada "Roomba Open Interface".

Nosotros usaremos esas funciones “creativas” con un módulo bluetooth para controlar el Roomba desde nuestra app.



# Capítulo 4

## Desarrollo del proyecto

### 4.1 Streaming

La primera parte que realicé fue la del streaming. Para ello comencé instalando el software de Wowza Streaming Engine, que se usa para configurar el servidor local, la dirección a la que tienes que hacer referencia en la app, y dar permiso a un usuario con contraseña para acceder al servidor.



Wowza Streaming Engine MANAGER

**License Expired.** Your Trial license has expired. Click [Buy Now](#) to purchase a license key for Wowza Streaming Engine™ software.

### Welcome to Wowza Streaming Engine!

Trial License - Expired on sep 3, 2018

**Status**

**Connections** Incoming and outgoing

**Usage** CPU, Memory, Heap and Disk

**Server Uptime**  
Since 30 Aug 2018 11:29:04 AM

**Features**  
Transcoder: Licensed  
DRM: Licensed  
nDVR: Licensed

**Test Video**  
To play an on demand test video, click [Test Players](#).

Wowza Streaming Engine MANAGER

### Source Authentication

Set up authentication to help secure RTMP-based and RTSP-based source connections to the server.

**Sources**

Name	Actions
user	<a href="#">Edit</a> <a href="#">Delete</a>

**Authenticating Live Sources**

By default, live applications in a Wowza Streaming Engine instance require that RTMP-based and RTSP-based sources be authenticated before they can connect and publish a live stream. Use this page to store user names and passwords for these sources. To manage user name/password authentication for incoming RTMP and RTSP source connections for an application, go to the application's **Source Security** page.

To view source account settings, click the source name in the list.

To create an account for a source, click the **Add Source** button.

To change source account settings, click the **Edit** icon for the source.

To delete a source account, click the **Delete** icon for the source.

**Ready to Purchase?**  
[Buy Now](#)

**Application Connection Settings**

Use the following settings to publish a stream to the Wowza Streaming Engine server.

Host - Server	169.254.169.189
Host - Port	1935
Application	A live application name on this server
Stream Name	The stream name you want to use
Login	A valid source user name and password

Wowza Streaming Engine MANAGER

### live > Monitoring

All dates/times shown are in your local browser time

**Connections Per Protocol** Incoming and outgoing

Protocol	Connections
Total:	1
Adobe HDS:	0
Adobe RTMP:	0
Apple HLS:	0
Microsoft Smooth:	0
MPEG DASH:	0
RTSP/RTMP:	1

**Application Uptime**  
live up since 03 Sep 2018 12:52:16 AM  
live up for about a minute

**Network Throughput**  
Bytes In: 2170295 @ 238.480 Kbits/s  
Bytes Out: 0 @ 0.000 Kbits/s

**Connections**

Select or enter date/time range

From: 02 Sep 2018 11:53 PM To: 03 Sep 2018 12:53 AM

☒ RTMP ☒ HDS ☒ DASH ☒ HLS ☒ RTSP/RTMP ☒ Smooth

☐ min ☐ max ☒ average ☐ actual

Una vez hecho esto, tenemos que incluir una librería en nuestro proyecto, llamada *libstreaming*, que es la que nos va a permitir controlar el streaming desde la parte de Android. Aquí se pueden configurar parámetros de la retransmisión de video, como el formato o el códec. También las variables de sesión con la clase *Session*:

```
public class Session {

    public final static String TAG = "Session";

    public final static int STREAM_VIDEO = 0x01;

    public final static int STREAM_AUDIO = 0x00;

    /** Some app is already using a camera (Camera.open() has failed). */
    public final static int ERROR_CAMERA_ALREADY_IN_USE = 0x00;

    /** The phone may not support some streaming parameters that you are trying to use (bit rate, fr
    public final static int ERROR_CONFIGURATION_NOT_SUPPORTED = 0x01;

}

/**
 * The internal storage of the phone is not ready.
 * libstreaming tried to store a test file on the sdcard but couldn't.
 * See H264Stream and AACStream to find out why libstreaming would want to something like that.
 */
public final static int ERROR_STORAGE_NOT_READY = 0x02;


/** The phone has no flash. */
public final static int ERROR_CAMERA_HAS_NO_FLASH = 0x03;

/** The supplied SurfaceView is not a valid surface, or has not been created yet. */
public final static int ERROR_INVALID_SURFACE = 0x04;

}

/**
 * The destination set with (@link Session#setDestination(String)) could not be resolved.
 * May mean that the phone has no access to the internet, or that the DNS server could not
 * resolved the host name.
```

Debemos configurar un archivo del proyecto en Android Studio, *AppConfig.java*, con la dirección del servidor y el usuario con la contraseña que hayamos asignado.



```
1 package com.jjoe64.motiondetection;
2
3 public class AppConfig {
4     public static final String STREAM_URL = "rtsp://192.168.1.34:1935/live/android_test";
5     public static final String PUBLISHER_USERNAME = "user";
6     public static final String PUBLISHER_PASSWORD = "1234";
7 }
8
```

Esas son las variables que usamos en la clase principal del proyecto para iniciar sesión de manera segura en el servidor de Wowza. A parte podemos guardar estos datos con el resto de datos de la app en el fichero de preferencias, para que queden guardados por si queremos entrar en el futuro no tener que logearte siempre.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Inicializamos la database y las variables con los datos del dispositivo
    database = new DatabaseFirebase( contexto: this);
    device name = Build.MANUFACTURER + " " + Build.MODEL;
    device id = Settings.Secure.getString(this.getContentResolver(), Settings.Secure.ANDROID_ID);
    deviceData = new DeviceData();

    //OBTENER LOS DATOS PARA EL STREAMING y ESTABLECERLOS EN EL APPCONFIG
    try {
        Bundle b = getIntent().getExtras();
        AppConfig.STREAM_URL = b.getString( key: "url");
        AppConfig.PUBLISHER_USERNAME = b.getString( key: "user");
        AppConfig.PUBLISHER_PASSWORD = b.getString( key: "password");

        //Guardar datos junto con el resto de datos de la app en el fichero preferences
        SharedPreferences myPreferences = PreferenceManager.getDefaultSharedPreferences( context: MainActivity.this);
        SharedPreferences.Editor myEditor = myPreferences.edit();
        myEditor.putString("url", AppConfig.STREAM_URL);
        myEditor.putString("user", AppConfig.PUBLISHER_USERNAME);
        myEditor.putString("password", AppConfig.PUBLISHER_PASSWORD);
        myEditor.apply();
        myEditor.commit();

    } catch (Exception e){
        Log.e( tag: "Error", e.getMessage());
    }
}

```

Una vez configurados los parámetros de inicio de sesión, tenemos que llamar a la función principal, *initRtspClient*, con la cual comienza el streaming, usando el surfaceview que declaramos en el layout principal, siendo el video encoder *H264*.

```

private void initRtspClient() {

    // Configures the SessionBuilder
    mSession = SessionBuilder.getInstance()
        .setContext(getApplicationContext())
        .setAudioEncoder(SessionBuilder.AUDIO_NONE)
        .setAudioQuality(new AudioQuality( samplingRate: 8000, bitRate: 16000))
        .setVideoEncoder(SessionBuilder.VIDEO_H264)
        .setSurfaceView(mSurfaceView).setPreviewOrientation(0)
        .setCallback(this).build();

    // Configures the RTSP client
    mClient = new RtspClient();
    mClient.setSession(mSession);
    mClient.setCallback(this);
    mSurfaceView.setAspectRatioMode(SurfaceView.ASPECT_RATIO_PREVIEW);
    String ip, port, path;

    // We parse the URI written in the EditText
    Pattern uri = Pattern.compile("rtsp://(.+):(\\d+)/(.+)");

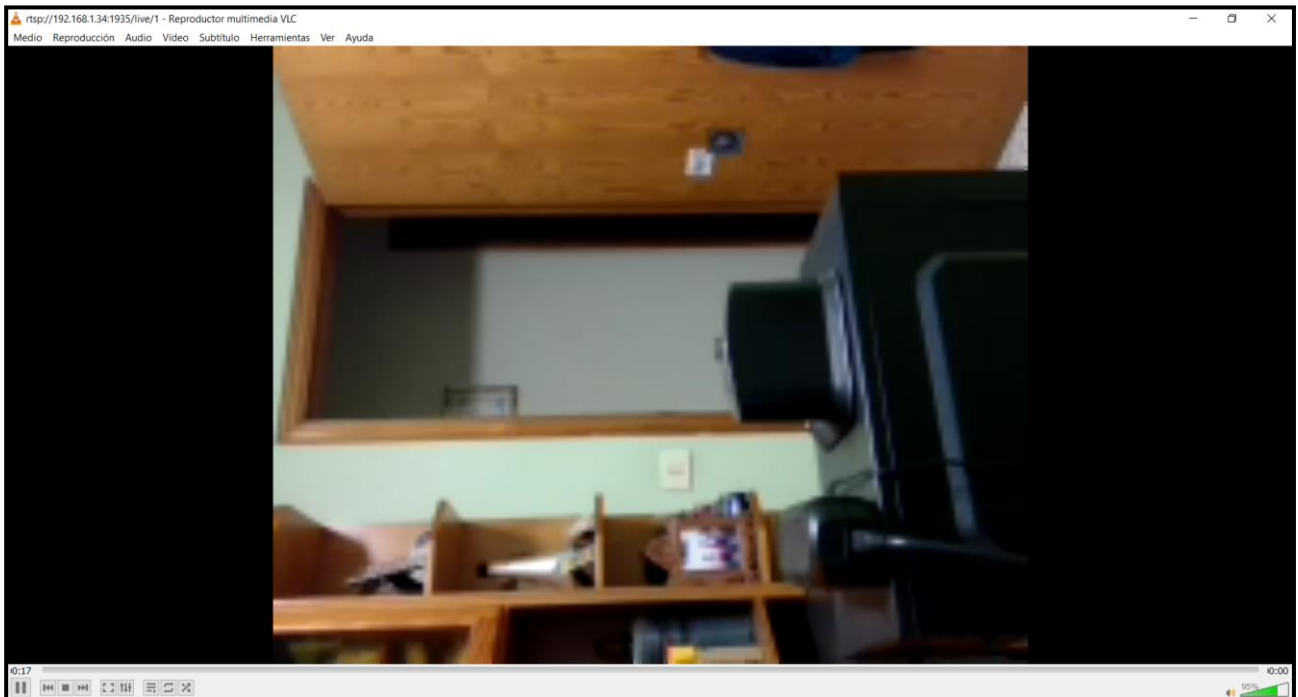
    SharedPreferences myPreferences = PreferenceManager.getDefaultSharedPreferences( context: this);
    Matcher m = uri.matcher(myPreferences.getString( key: "url", defValue: "null"));

    m.find();
    ip = m.group(1);
    port = m.group(2);
    path = m.group(3);

    mClient.setCredentials(myPreferences.getString( key: "user", defValue: "null"), myPreferences.getString( key: "password", defValue: "null"));
    mClient.setServerAddress(ip, Integer.parseInt(port));
    mClient.setStreamPath("/") + path;
}

```

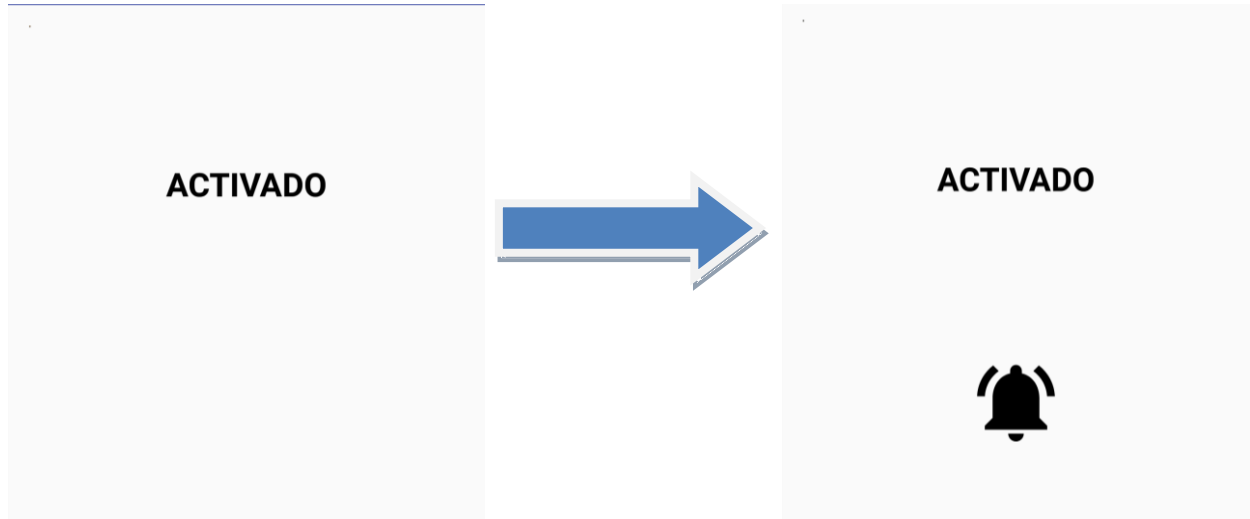
Cuando configuramos estos parámetros, y antes de tener la web con el servidor activa, podemos confirmar que el streaming funciona con el reproductor VLC. En la parte de arriba del programa se puede ver la dirección web que se le ha asignado (*rtsp://192.168.1.34:1935/live/1*). El *1* del final es un identificador, en caso de haber más cámaras haciendo streaming a la vez, debería ir aumentando (*rtsp://192.168.1.34:1935/live/*, *rtsp://192.168.1.34:1935/live/3*, etc).





## 4.2 Acople de detección de movimiento

Para este paso hizo falta crear primero una app sencilla que hiciera la detección de movimiento, con una interfaz simple que cuando detectaba movimiento hacia visible un botón, para comprobar que funcionaba.



La función trabaja comparando frames en un intervalo de tiempo regulable para hacer que sea más o menos sensible a los movimientos. También podemos regular la sensibilidad de los colores en una de las clases de la librería usadas para este módulo de la aplicación.

```
public static int[] decodeYUV420SPtoRGB(byte[] yuv420sp, int width, int height) {
    if (yuv420sp == null) throw new NullPointerException();

    final int frameSize = width * height;
    int[] rgb = new int[frameSize];

    for (int j = 0, yp = 0; j < height; j++) {
        int uvp = frameSize + (j >> 1) * width, u = 0, v = 0;
        for (int i = 0; i < width; i++, yp++) {
            int y = (0xff & (yuv420sp[yp])) - 16;
            if (y < 0) y = 0;
            if ((i & 1) == 0) {
                v = (0xff & yuv420sp[uvp++]) - 128;
                u = (0xff & yuv420sp[uvp++]) - 128;
            }
            int y1192 = 1192 * y;
            int r = (y1192 + 1634 * v);
            int g = (y1192 - 833 * v - 400 * u);
            int b = (y1192 + 2066 * u);

            if (r < 0) r = 0;
            else if (r > 262143) r = 262143;
            if (g < 0) g = 0;
            else if (g > 262143) g = 262143;
            if (b < 0) b = 0;
            else if (b > 262143) b = 262143;

            rgb[yp] = 0xff000000 | ((r << 6) & 0xff0000) | ((g >> 2) & 0xff00) | (
        }
    }
    return rgb;
}
```

```
private final AggregateLumaMotionDetection detector;
private long checkInterval = 500;
private long lastCheck = 0;
private MotionDetectorCallback motionDetectorCallback;
private Handler mHandler = new Handler();
```



Usa una función *isDifferent* donde se comparan los frames y devuelve un booleano *true* o *false*, se compara frame a frame, usando las clases de la librería.

```
public boolean isDifferent(State s1, State s2) {
    if (s1 == null || s2 == null) throw new NullPointerException();
    if (s1.getWidth() != s2.getWidth() || s1.getHeight() != s2.getHeight()) return true;

    // Boxes
    this.variance = new int[yBoxes][xBoxes];

    // set to a different by default, if a change is found then flag
    // non-match
    boolean different = false;
    // loop through whole image and compare individual blocks of images
    int b1 = 0;
    int b2 = 0;
    int diff = 0;
    for (int y = 0; y < yBoxes; y++) {
        for (int x = 0; x < xBoxes; x++) {
            b1 = aggregateMapArea(state1.getMap(), x, y);
            b2 = aggregateMapArea(state2.getMap(), x, y);
            diff = Math.abs(b1 - b2);
            variance[y][x] = diff;
            // the difference in a certain region has passed the threshold
            // value
            if (diff > leniency) different = true;
        }
    }
    return different;
}
```

Inclusive tiene una función que detecta si hay mucha oscuridad como para hacer la comparativa de frames, basándonos en la escala RGB, si es cada uno de los valores 0, es que no se está detectando ningún color, por lo tanto es todo oscuridad.

Too dark here

```
while (isRunning.get()) {
    long now = System.currentTimeMillis();
    if (now - lastCheck > checkInterval) {
        lastCheck = now;

        if (nextData.get() != null) {
            int[] img = ImageProcessing.decodeYUV420SPtoLuma(nextData.get(), nextWidth.get(), nextHeight.get());

            // check if it is too dark
            int lumaSum = 0;
            for (int i : img) {
                lumaSum += i;
            }
            if (lumaSum < minLuma) {
                if (motionDetectorCallback != null) {
                    mHandler.post(() -> {
                        motionDetectorCallback.onTooDark();
                    });
                }
            } else if (detector.detect(img, nextWidth.get(), nextHeight.get())) {
                // check
                if (motionDetectorCallback != null) {
                    mHandler.post(() -> {
                        motionDetectorCallback.onMotionDetected();
                    });
                }
            }
        }
    }
    try {
        Thread.sleep(millis: 10);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Una vez comprobado que funcionaba y establecidos unos parámetros acordes a lo que pretendíamos buscar, se pasó a integrar en segundo plano para no afectar así al funcionamiento del streaming.

Para finalizar este paso se procedió a cambiar la forma de configurar la ruta y el usuario de acceso al servicio de streaming, para hacerlo en dinámico desde la propia app. Para ello se cambió la interfaz, añadiendo una nueva pantalla o *intent*, en el que se recoge la *URL*, el *Usuario* y la *Contraseña*. También se añadió un acceso directo a los permisos de la aplicación, debido a que esta misma necesita permisos de escritura, de internet, y acceso a la cámara y al micrófono.

Configuración

URL  
rtsp://192.168.1.34:1935/live/1

Usuario  
user

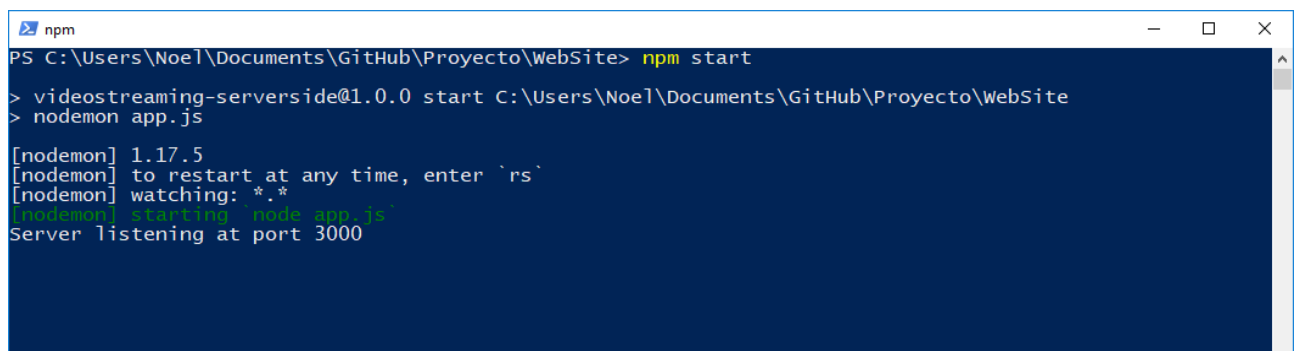
Contraseña  
• • • •

COMENZAR STREAMING

ESTABLECER PERMISOS

## 4.3 Comunicación con el servidor web

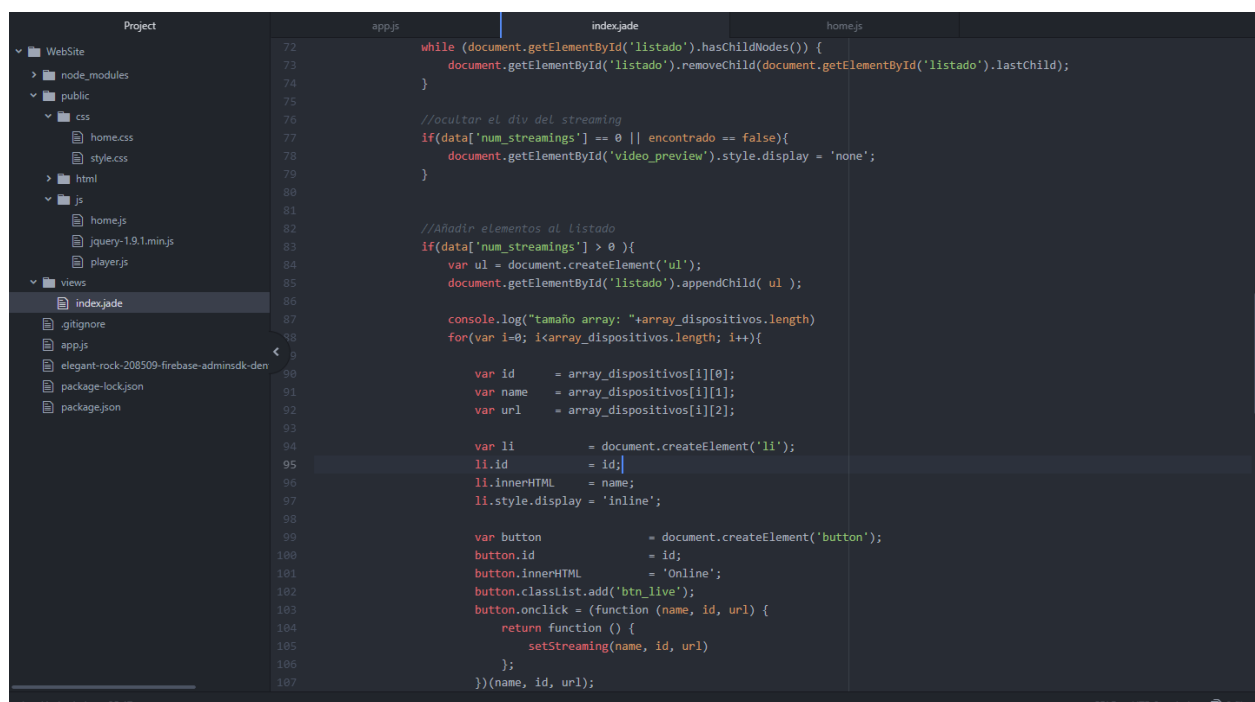
En este paso fue necesario el uso del editor de textos *Atom* y de una *PowerShell* de Windows 10 para ejecutar los comandos de node.js. Lo primero fue crear la estructura de directorios con sus correspondientes ficheros para el funcionamiento de la web. Usaremos también npm, que es el manejador de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript. Con el comando *npm start* levantamos el servidor en localhost, usando el puerto que indicamos en el código. Lo que nos permite este comando es no tener que parar y relanzar el servidor cada vez que realicemos un cambio, puesto que detecta cuando se guarda un archivo para volver a reiniciarlo.



```
PS C:\Users\Noel\Documents\GitHub\Proyecto\WebSite> npm start
> videostreaming-serverside@1.0.0 start C:\Users\Noel\Documents\GitHub\Proyecto\WebSite
> nodemon app.js

[nodemon] 1.17.5
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting node app.js
Server listening at port 3000
```

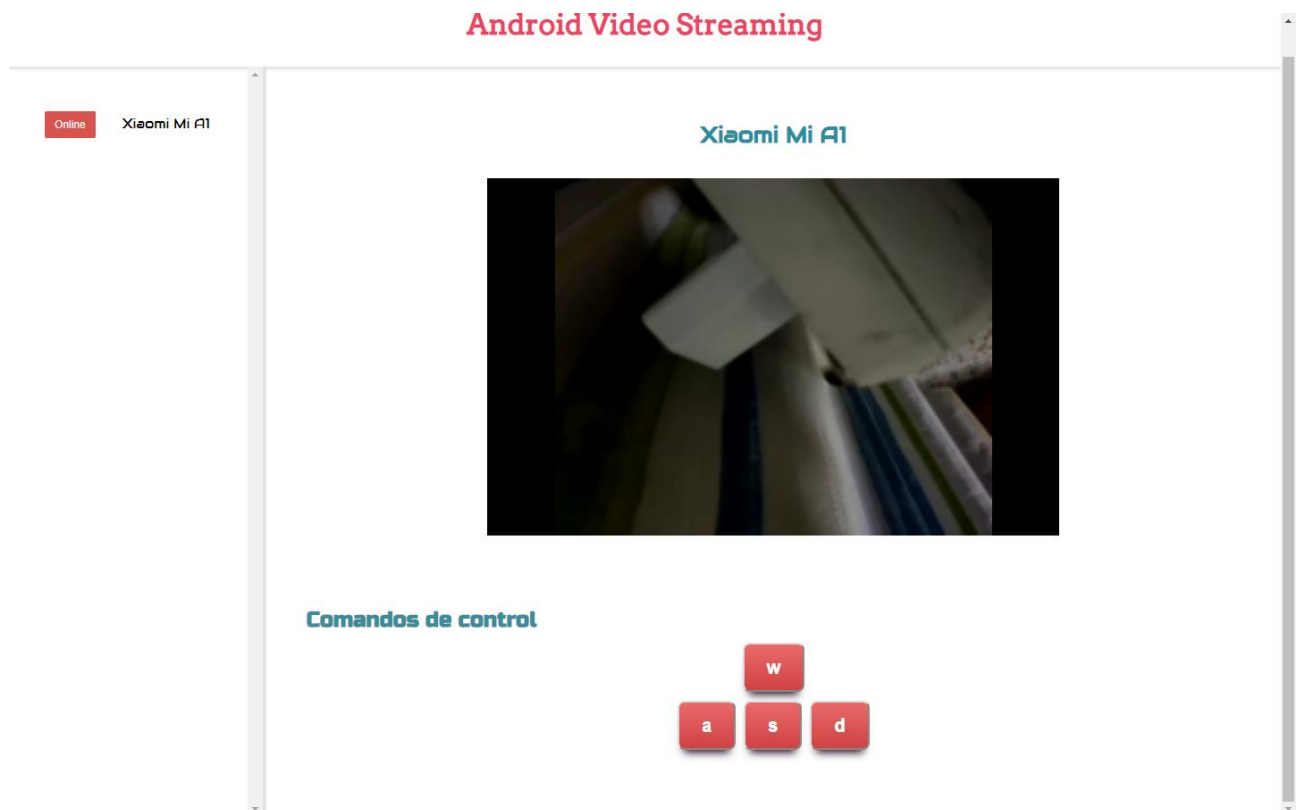
La parte del front-end fue desarrollada con jade, generando el código html mediante plantillas. Se trataba de crear un panel de control sencillo e intuitivo pero eficaz. Los datos para generarlo se pasan mediante un socket, que son el nombre del dispositivo desde el cual se está haciendo streaming y el estado del streaming. Si esta en activo, se vuelve visible en la interfaz y se puede acceder al streaming. Si no permanece oculto hasta que se active el streaming en el dispositivo.



```
Project
  WebSite
    node_modules
    public
    css
      home.css
      style.css
    html
    js
      home.js
      jquery-1.9.1.min.js
      player.js
    views
      indexjade
      .gitignore
      app.js
      elegant-rock-208509-firebase-adminsdk-den
      package-lock.json
      package.json

72 while (document.getElementById('listado').hasChildNodes()) {
73   document.getElementById('listado').removeChild(document.getElementById('listado').lastChild);
74 }
75
76 //ocultar el div del streaming
77 if(data['num_streamings'] == 0 || encontrado == false){
78   document.getElementById('video_preview').style.display = 'none';
79 }
80
81
82 //Añadir elementos al listado
83 if(data['num_streamings'] > 0){
84   var ul = document.createElement('ul');
85   document.getElementById('listado').appendChild( ul );
86
87   console.log("tamaño array: "+array_dispositivos.length)
88   for(var i=0; i<array_dispositivos.length; i++){
89
90     var id      = array_dispositivos[i][0];
91     var name    = array_dispositivos[i][1];
92     var url     = array_dispositivos[i][2];
93
94     var li      = document.createElement('li');
95     li.id       = id;
96     li.innerHTML = name;
97     li.style.display = 'inline';
98
99     var button  = document.createElement('button');
100    button.id    = id;
101    button.innerHTML = 'Online';
102    button.classList.add('btn_live');
103    button.onclick = (function (name, id, url) {
104      return function () {
105        setStreaming(name, id, url)
106      };
107    })(name, id, url);
```

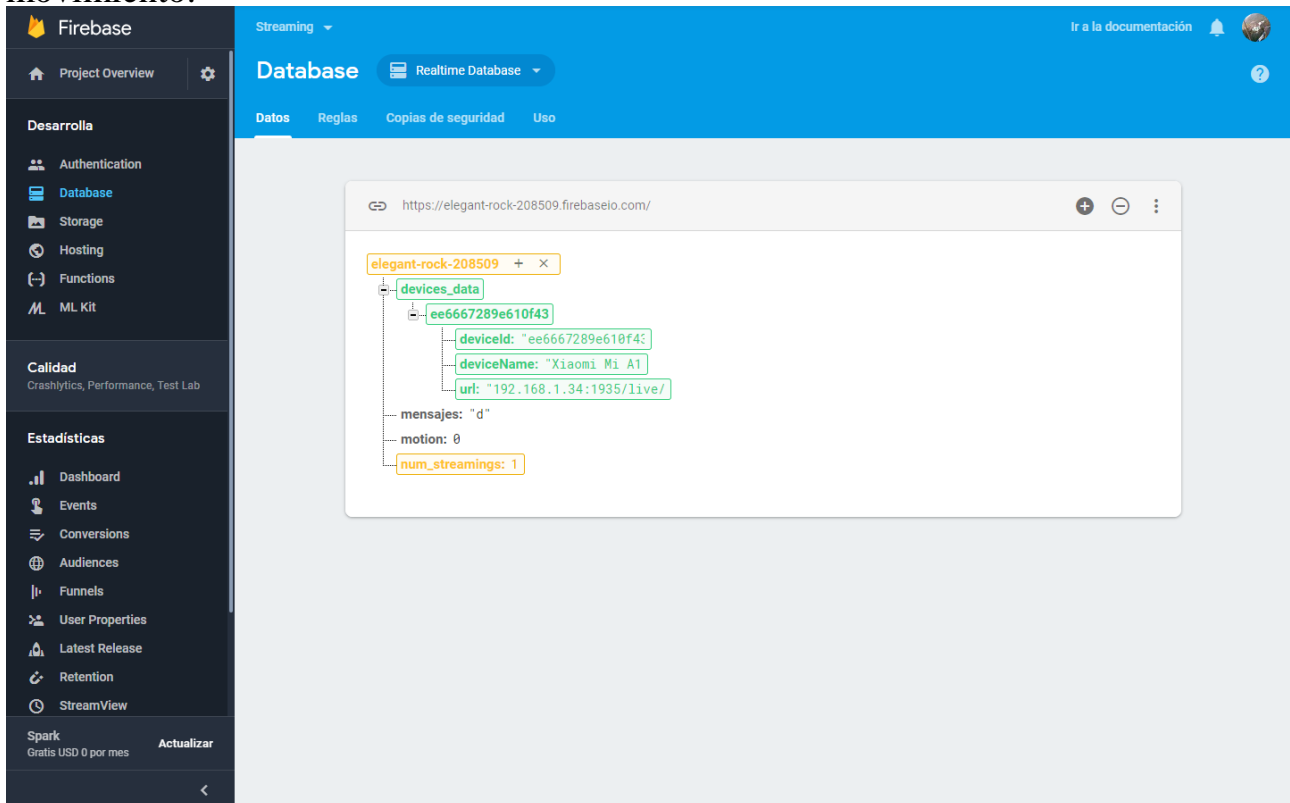
Aparte de esto, se incorporó también un sistema de control del Roomba, con cuatro sencillos movimientos para enviar mediante un módulo wifi o bluetooth del que hablaremos más adelante.



Cabe destacar que el servidor node.js funciona mediante el modelo vista-controlador, separando los datos y la lógica de la aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Esto se ve reflejado en la estructura de los directorios.

## 4.4 Integración con Firebase

Para dar seguridad y una respuesta en tiempo real, sin necesidad de abrir un socket entre la app de Android y el servidor node.js, aparece la opción de usar esta plataforma. Hacemos uso de su base de datos en tiempo real, añadiendo los valores que hacen falta, como el nombre y el ID del dispositivo, la URL del streaming, el número de streamings actual, el movimiento para la Roomba y si hay o no movimiento.



Para integrarlo en la aplicación de Android Studio debemos seguir una serie de pasos.

Lo primero es crear un proyecto en *Firebase Console*, lo agregamos a nuestra cuenta y le cambiamos el ID si es necesario. Una vez creado podemos ver la descripción de nuestro proyecto.

Luego de esto debemos descargar un archivo JSON, que se puede descargar en cualquier momento en caso de querer agregarlo a otro proyecto o en caso de pérdida.

El siguiente paso es agregar el SDK, para ello debemos incorporar a nuestro archivo *build.gradle* del proyecto en Android Studio el complemento de google-service y el repositorio maven.

```

buildscript {
    // ...
    dependencies {
        // ...
        classpath 'com.google.gms:google-services:4.0.1' // google-services plugin
    }
}

allprojects {
    // ...
    repositories {
        // ...
        google() // Google's Maven repository
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.android.support:support-v4:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.2'
    implementation 'com.github.nkzawa:socket.io-client:0.5.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.'
    implementation project(path: ':libstreaming')
    implementation 'com.google.firebase:firebase-core:16.0.1'
    implementation 'com.google.firebase:firebase-database:16.0.1'
    implementation 'com.google.code.gson:gson:2.8.5'
}

```

A partir de aquí debemos desarrollar el código en el IDE Android Studio.

En una nueva clase del proyecto se crean las funciones con las que se controla la base de datos. Por ejemplo, aquí mostramos dos de ellas, en la primera se leen los datos del dispositivo y se pasan a la base de datos en las variables que se declaran, puesto que las únicas entradas de la base de datos son el número de streamings, los comandos para mover el Roomba y si ha detectado movimiento el streaming actual. El resto (el árbol que se crea a partir de un ID de cualquier dispositivo) y queda colgando de una rama padre, que es eliminada cuando termina el streaming.

```

public class DeviceData {

    private String device_id, device_name, url;
    public int motion;

    public DeviceData() {
    }

    //getter y setter
    public void setDeviceId(String device_id) { this.device_id = device_id; }

    public void setDeviceName (String device_name) { this.device_name = device_name; }

    public void setUrl (String url) { this.url = url.substring(7); }

    public String getDeviceId() { return device_id; }

    public String getDeviceName() { return device_name; }

    public String getUrl() { return url; }

    public void setMotion(int motion) {this.motion = motion;}

    public int getMotion() {return motion;}

}

```

```

public void setDeviceData(String device_name, String device_id, String url, int mo){
    DatabaseReference myRef = database.getReference("devices_data");

    DeviceData data = new DeviceData();
    data.setDeviceId(device_id);
    data.setDeviceName(device_name);
    data.setUrl(url);
    data.setMotion(mo);
    myRef.child(device_id).setValue(data);
}

public void removeDeviceData(String device_id){
    DatabaseReference myRef = database.getReference("devices_data");
    myRef.child(device_id).removeValue();
}

```

En el activity principal de la aplicación tenemos que hacer la llamada a estas funciones, puesto que es de donde obtenemos el nombre del dispositivo, su ID y la URL del servidor Wowza, aparte de resetear el detector de movimiento. Esta llamada se hace dentro de la función *initRtspClient* que ya definimos antes.

```

// Configures the RTSP client
mClient = new RtspClient();
mClient.setSession(mSession);
mClient.setCallback(this);
mSurfaceView.setAspectRatioMode(SurfaceView.ASPECT_RATIO_PREVIEW);
String ip, port, path;

// We parse the URI written in the EditText
Pattern uri = Pattern.compile("rtsp://(.+):(\\d+)/(.+)");

SharedPreferences myPreferences = PreferenceManager.getDefaultSharedPreferences(context, this);
Matcher m = uri.matcher(myPreferences.getString(key: "url", defValue: "null"));

m.find();
ip = m.group(1);
port = m.group(2);
path = m.group(3);

mClient.setCredentials(myPreferences.getString(key: "user", defValue: "null"), myPreferences.getString(key: "password", defValue: "null"));
mClient.setServerAddress(ip, Integer.parseInt(port));
mClient.setStreamPath("/") + path;

// Sumamos uno al contador de streamings al conectarse por rtsp
database.getAndModifyNumStreaming(operation: "+");

database.setDeviceData(device_name, device_id, AppConfig.STREAM_URL, mo: 0);
}

```

En la parte del servidor también tenemos que hacer cambios, en el archivo *app.js* tenemos que añadir la función *manageDatabase*, con la que traeremos la base de datos de Firebase cuando detecte cambios para mostrarlos en la web. Usamos dentro de la función *manageDatabase* la función *child\_added* para crear la nueva entrada, con los datos que traemos de la base de datos.

```

manageDatabase();
function manageDatabase(){
    ref.on("value", function(snapshot) {
        var cont = 0 ;
        numstreaming = snapshot.val();
        var arr = new Array();
        var array_dispositivos = new Array();

        refr.on("child_added", function(snapshot, prevChildKey) {
            cont ++;
            if(cont == 1){
                snapshot.forEach(function (android_id) {
                    device_id = android_id.val().deviceId;
                    device_name = android_id.val().deviceName;
                    url = android_id.val().url;
                    motion = android_id.val().motion;
                    arr = [device_id, device_name, url, motion];
                    array_dispositivos.push(arr);
                });
            }

            if(socket_io != undefined){
                console.log("socket_io.emit 1")
                socket_io.emit('data_streaming',{'num_streamings': numstreaming, 'dispositivos': array_dispositivos});
            }
        });
    });
}

```

Lo mismo para eliminar un dispositivo cuando termine su streaming. Usamos la función *child\_removed*, así eliminamos esa entrada de la web y todos los datos asociados a ella.

```

refr.on("child_removed", function(snapshot) {
    cont--;
    console.log("socket_io.emit 2")
    snapshot.forEach(function (android_id) {
        device_id = android_id.val().deviceId;
        device_name = android_id.val().deviceName;
        url = android_id.val().url;
        motion = android_id.val().motion;
        arr = [device_id, device_name, url, motion];
        array_dispositivos.push(arr);
    });

    if(socket_io != undefined){
        console.log("socket_io.emit 2")
        socket_io.emit('data_streaming',{'num_streamings': numstreaming, 'dispositivos': array_dispositivos});
    }
});

```



# Capítulo 5

## Problemas encontrados

### 5.1 Detección de movimiento y streaming

A la hora de integrar estas dos aplicaciones, que en un principio estaban separadas, se encontraron algunos problemas.

El primero fue directamente la incompatibilidad de compartir el mismo *MainActivity* para ejecutarlas una sobre la otra (en primer plano el streaming, en segundo plano la detección de movimiento). En el *OnResume* de este activity no permitía iniciar el streaming y la detección a la vez.

La solución encontrada fue iniciar solo la detección de movimiento en el *OnResume*, junto con las variables de sesión y los parámetros para iniciar el streaming, pero sin llegar a iniciarlo en el *SurfaceView* del layout. Este paso lo realizaremos después, cuando este lo demás iniciado.

```
@Override
protected void onResume() {
    super.onResume();

    motionDetector.onResume();

    // Start camera preview
    mSession.startPreview();

    // Start video stream
    mClient.startStream();
    //toggleStreaming();
    database.getAndModifyNumStreaming( operation: "+");
}
```

```
motionDetector = new MotionDetector( context: this, (android.view.SurfaceView) findViewById(R.id.surfaceView1));
Log.d( tag: "motion", msg: "antes");
motionDetector.setMotionDetectorCallback(new MotionDetectorCallback() {

    @Override
    public void onMotionDetected() {
        Log.d( tag: "motion", msg: "si");
        contador++;
        if(contador >= 2){
            database.setDeviceData(device_name, device_id, AppConfig.STREAM_URL, mo: 1);
            database.setMotionGeneral(1);
        }
    }

    @Override
    public void onTooDark() {
        //txtStatus.setText("Too dark here");
        Toast.makeText( context: MainActivity.this, text: "Oscuro", Toast.LENGTH_SHORT).show();
    }
});
Log.d( tag: "motion", msg: "sale");

mSurfaceView = (SurfaceView) findViewById(R.id.surface);
mSurfaceView.getHolder().addCallback(this);

// Initialize RTSP client
initRtspClient();
```

## 5.2 Socket de comunicación servidor

Para comunicarnos entre la aplicación y la aplicación necesitábamos abrir un socket de comunicación. El problema era la falta de seguridad, y la necesidad de estar en la misma red tanto el dispositivo que ejecutaba la aplicación como el servidor web en node.js.

En un principio se estudió la posibilidad de abrir un socket bidireccional, para indicar el nombre del dispositivo, el ID e informar de la detección de movimiento. El problema es que no era suficiente para hacer todas las tareas, incluida la de enviar los mensajes de movimiento para la Roomba. Los mensajes de movimiento llegaban, pero no era en tiempo real, ni era seguro, ni funcionaba en distintas redes.

A parte tampoco notificaba la detección de movimiento, pues no se encontró la manera de que siguiera abierto en segundo plano sin tener que cerrar alguna de las comunicaciones en un sentido u otro.

La solución a esto fue usar Firebase como base de datos en tiempo real para comunicar el servidor con la aplicación, y solucionar así todos estos problemas de un solo golpe.

Firebase permitía que las conexiones fueran seguras, ya que cuentan con el estándar de seguridad de google, que fueran en tiempo real, que es una de las principales características que presenta Firebase, y poder comunicar los mensajes sin tener que parar la escucha para la detección de movimiento.

## 5.3 Repetición dispositivos

A la hora de añadir un dispositivo, la función `ref.on("value", function(snapshot){})`, que debía llamarse una única vez por cada vez que se modificase el valor del `ref`, se estaba llamando multiples veces por cada modificación que se realizaba sobre la referencia.

Esta función estaba siendo utilizada para listar los dispositivos conectados al streaming, y con este problema se listaba el mismo dispositivo varias veces.

La solución fue usar un filtro para controlar que si un dispositivo estaba ya en la lista de dispositivos en streamings, no volviera a añadirlo.

## 5.4 Reproductor del streaming

El problema viene que no todos los reproductores que soporta un navegador web permitía el Real Time Messaging Protocol, y antes de dar con uno que si lo aceptaba, se usaba otro reproductor.

A la hora de hacer pruebas con el servidor de Wowza y no mostrar nada en pantalla se pensó que el error venia del propio servidor o incluso de la aplicación Android, y fue una vez descartado las dos opciones anteriores cuando se dio con el problema.

El reproductor anterior que estaba en uso no soportaba RTMP, así que se optó por usar *Videojs*, un reproductor que si permitia ese protocolo. A parte el reproductor debía permitir el uso de flash.

Otro inconveniente que surgió era que para cada dispositivo necesitábamos un streaming diferente, el que le correspondía a cada dispositivo, y si incluíamos estáticamente el reproductor en HTML, la URL del streaming seria la misma para todos los dispositivos.

La solución fue añadir dinámicamente el reproductor por cada dispositivo que se registrara en la web y así cada streaming entrante tenia asignada su URL RTMP correcta.

# **Capítulo 6**

## **Conclusiones y resultados**

# **Capítulo 7**

## **Summary and Conclusion**

# Capítulo 8

## Presupuesto

Para calcular el presupuesto estimado se realizará un cálculo básico en función de las horas de trabajo necesarias y las herramientas que necesitamos.

### 8.1 Estimación coste de herramientas

En nuestro caso la mayoría de las herramientas que usamos son gratuitas, pero si necesitaremos un servidor para lanzar nuestra web, un dispositivo móvil para ejecutar la aplicación y una aspiradora Roomba.

Herramienta	Precio
Dominio web	100€/año
Dispositivo móvil	300€
Roomba	350€

### 8.2 Estimación coste de desarrollo e implantación de la aplicación

Para calcular el salario por hora, se toma como referencia salarios medios de desarrolladores web y Android en España, que oscila los 15€/hora según datos de INE. Dicho esto procedemos a desglosar en una tabla las horas necesarias y su coste.

Tareas	Horas	Precio
Detección de movimiento	20	300€
Streaming Wowza	12	180€
Integración aplicaciones	24	630€
Servidor web	8	120€
Socket Firebase	16	240€
Experimentación	30	450€

## 8.3 Presupuesto final estimado

Partes	Precio
Herramientas	850€
Desarrollo	1920€
<b>Precio Final</b>	<b>2770€</b>

El presupuesto final de la aplicación, tanto el coste de las herramientas como el coste de desarrollo asciende a **2770€**.

# **Capítulo 9**

## **Bibliografía**

<https://www.observatoriorh.com/mercado-de-trabajo/sueldo-desarrollador-web-ano-experiencia-superior-media-salarial-espanola.html>