# Proyecto DAPP


Arquitectura e Integración de Sistemas Software

Grado de Ingeniería del Software

Curso 2

Gonzalo Álvarez García (gonalvgar@alum.us.es)
Alfonso Cadenas Morales (alfcadmor@alum.us.es)
Guillermo Losada Ostos (guilosost@alum.us.es)
Miguel Yanes Ariza (migyanari@alum.us.es)

Tutor: Javier Troya Castilla
Número de grupo:

Enlace de la aplicación: https://project-dapp.appspot.com/

## HISTORIAL DE VERSIONES

| Fecha | Versión | Detalles | Participantes |
|---|---|---|---|
| 17/03/2019 | 1.0 | - Incluye introducción, prototipos de las interfaces de usuario y diagramas UML de componentes y despliegue. | Gonzalo Álvarez García<br>Alfonso Cadenas Morales<br>Guillermo Losada Ostos<br>Miguel Yanes Ariza |
| 28/04/2019 | 2.0 | - Incluye todo el contenido del anterior entregable actualizado a las nuevas APIs que se han implementado. Además, se añade la API y detalles de la implementación. | Gonzalo Álvarez García<br>Alfonso Cadenas Morales<br>Guillermo Losada Ostos<br>Miguel Yanes Ariza |
| 20/05/2019 | 3.0 | -Incluye todo el contenido del anterior entregable. Se actualiza nuestra API Rest, amplían las pruebas, detalles de implementación y todos los diagramas. | Gonzalo Álvarez García<br>Alfonso Cadenas Morales<br>Guillermo Losada Ostos<br>Miguel Yanes Ariza |

# Índice

# 1 Introducción

Esta aplicación tiene como objetivo realizar búsquedas simultáneas de imágenes y vídeos en el ámbito digital, reuniendo tres aplicaciones de image/video hosting service (YouTube, DeviantArt y Dailymotion) en un mashup. Otra de las funcionalidades que ofrece es la facilidad para ofrecer feedback a las publicaciones encontradas tras la búsqueda mediante valoraciones y comentarios.

Por último, se proporcionan las estadísticas de las cuentas con las que hayamos iniciado sesión, incluyendo datos como: el número de visitas, la publicación más vista, valoraciones totales, etc.

## 1.1 Aplicaciones integradas

Las tres aplicaciones que conforman el mash-up ofrecen servicios de redes sociales en las que se puede publicar de forma inmediata en forma de imágenes y vídeos.

Las tres aplicaciones tienen un formato muy similar, por lo que explicarlas una a una sería redundante. Básicamente, son plataformas de subida e intercambio de imágenes y/o vídeos online, donde las fotos y vídeos de los usuarios pueden ser sometidas a las críticas y opiniones de otros.

| Nombre aplicación | URL documentación API |
|---|---|
| DeviantArt | https://www.deviantart.com/developers/ |
| YouTube | https://developers.google.com/youtube/v3/ |
| Dailymotion | https://developer.dailymotion.com/api |

TABLA 1. APLICACIÓN INTEGRADAS

## 1.2 Evolución del proyecto

La idea inicial consistía en implementar los clientes de Twitter, Tumblr e Instagram dentro de nuestra aplicación y poder publicar tanto en todas las redes a la vez como en cada una individualmente. Por incompatibilidades en los servicios de autenticación, tuvimos que cambiar dos de las tres aplicaciones y añadir otra más, que serían las definitivas: Twitter, Facebook, Reddit y Pinterest – todas coinciden en el uso de OAuth2.

Además, descartamos implementar el cliente completo de todas las aplicaciones ya que es algo muy complejo y no mejoraría el uso de la aplicación. Por ello, finalmente decidimos que la aplicación solo publicaría en todas las redes e informaría de las estadísticas de feedback de las publicaciones realizadas.

Tras el cambio de aplicaciones del primer entregable, volvimos a encontrarnos con más problemas. En este caso, la problemática radicaba en los permisos que precisaban Facebook y Pinterest para poder usar sus servicios de publicación y recogida de datos, la autenticación híbrida de Twitter que mezclaba OAuth 1.0 y 2.0 y la obtención del

access token de Reddit, por lo que tuvimos que deshacer completamente el primer proyecto y empezar de cero otro nuevo.

Para este segundo proyecto intentamos implementar una aplicación de funcionalidad muy parecida a la anterior solo que cambiando el tipo de publicación que se hace, que pasa de ser en formato de texto a formato imagen, ya que solo utilizaríamos plataformas de image hosting – DeviantArt, Unsplash, Imgur, Flicker y Google Photos.

Sin embargo, nos volvimos a encontrar con problemas en las APIs seleccionadas previamente, de nuevo a la hora de obtener el access token, viéndonos forzados a cambiar las aplicaciones por las plataformas de publicación de imagen y vídeo YouTube, DeviantArt y Dailymotion. Aún así, sin alejarse mucho de la idea inicial, ya que el objetivo es poder hacer búsquedas y POSTs en estas plataformas además de obtener estadísticas del usuario autenticado.

# 2 Prototipos de interfaz de usuario

## 2.1 Vista inicio de sesión

Vista del inicio de sesión en las tres redes plataformas.

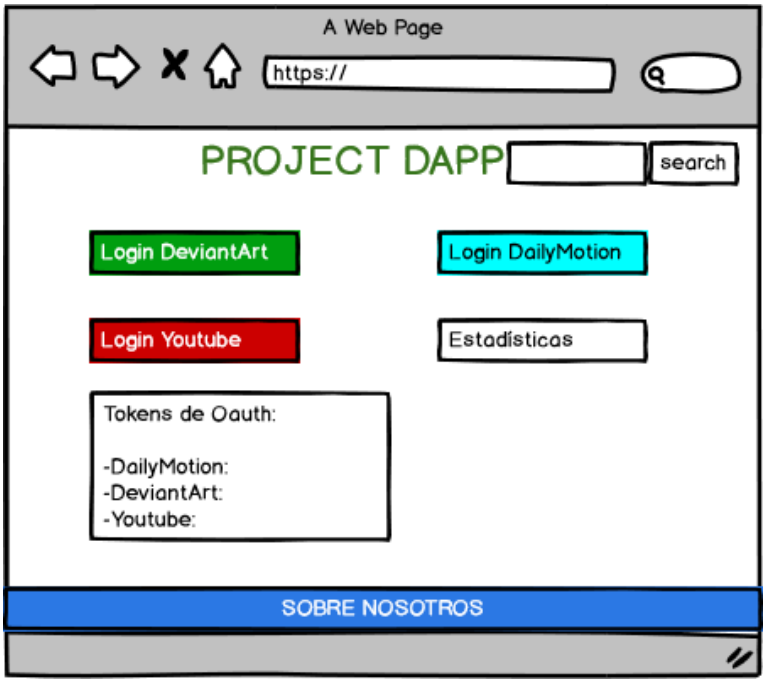



FIGURA 1. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA DE INICIO DE SESIÓN

## 2.2 Vista de búsqueda

Vista de la interfaz al realizar una búsqueda.

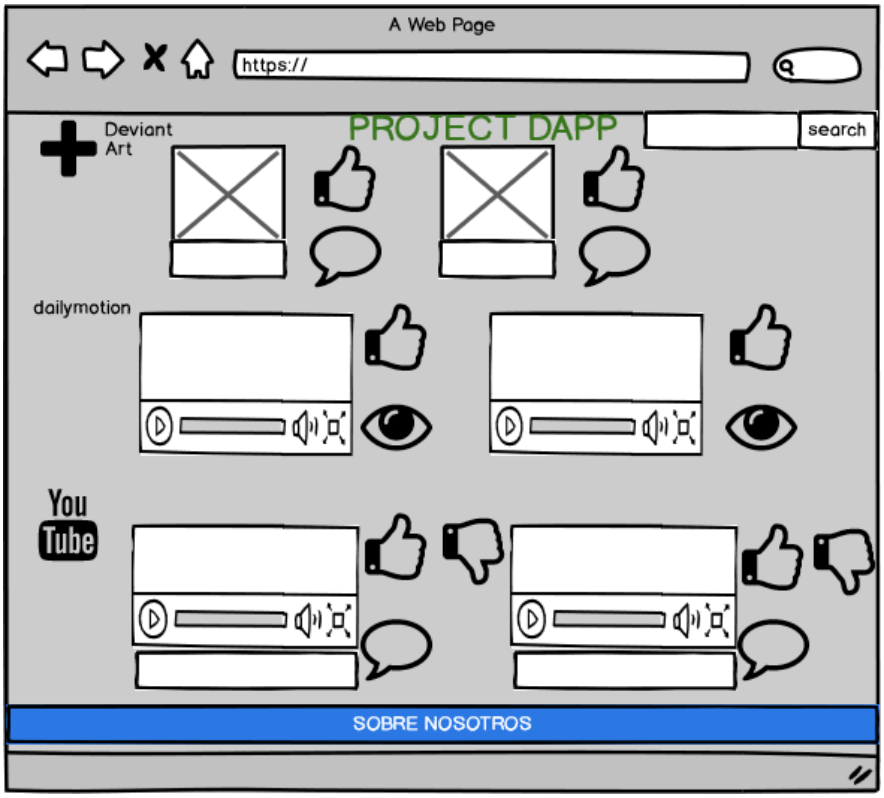



FIGURA 2. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA DE BÚSQUEDA

## 2.3 Vista de las estadísticas

Vista de las estadísticas de las publicaciones que se han hecho desde las diferentes plataformas para conocer el feedback.

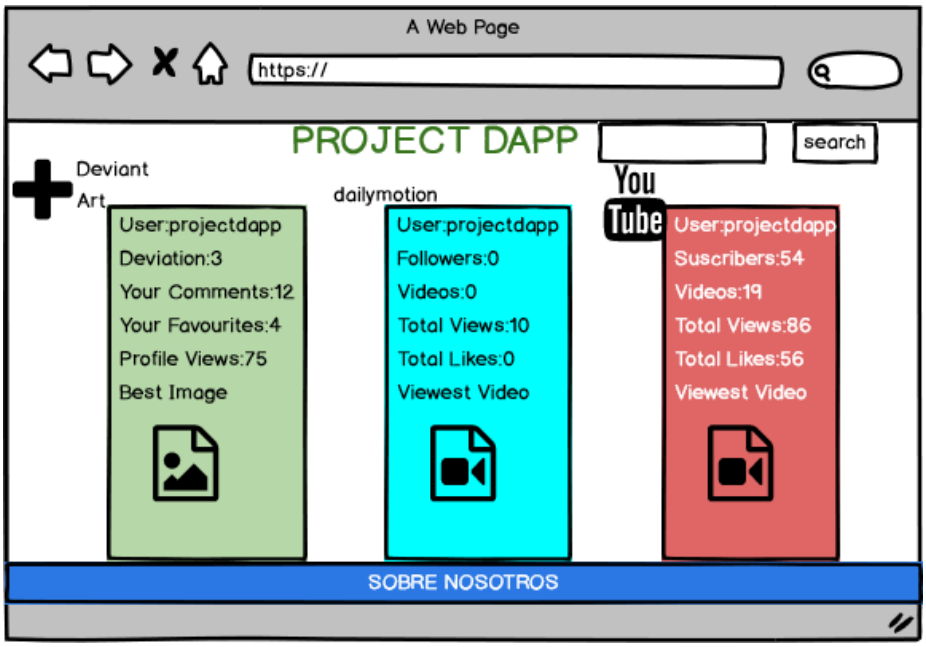
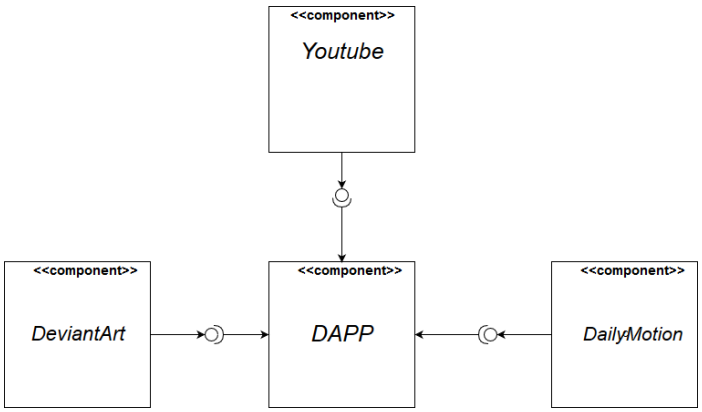


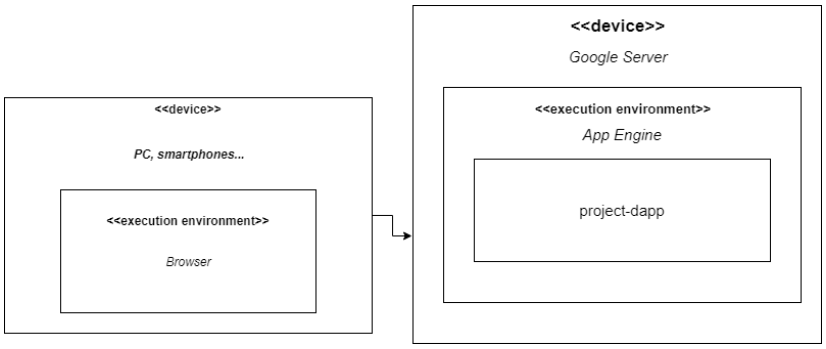

FIGURA 3. PROTOTIPO DE INTERFAZ DE USUARIO DE LA VISTA DE LAS ESTADÍSTICAS

# 3 Arquitectura

## 3.1 Diagrama de componentes




## 3.2 Diagrama de despliegue




## 3.3 Diagrama de secuencia de alto nivel

Diagrama de búsqueda de fotos/vídeos simultánea en todas las plataformas.

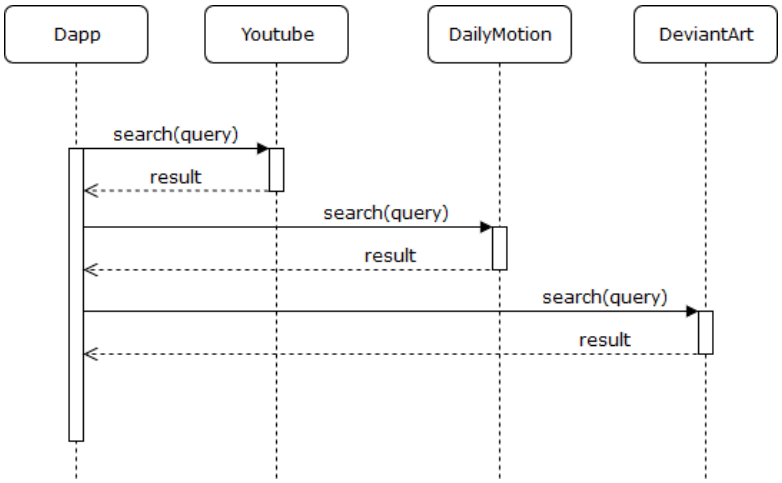
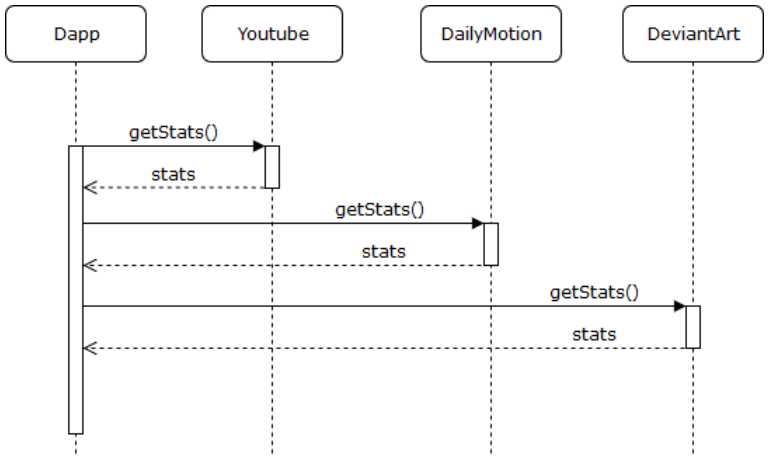


Diagrama de petición de las estadísticas de cada plataforma.




## 3.4 Diagrama de clases

Diagrama MVC SearchImages

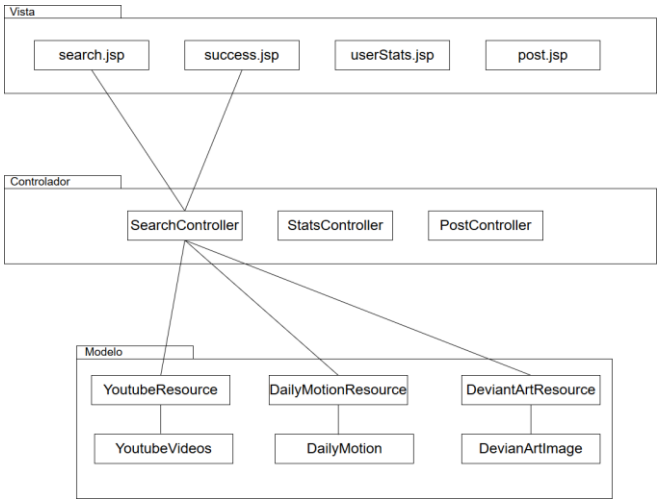
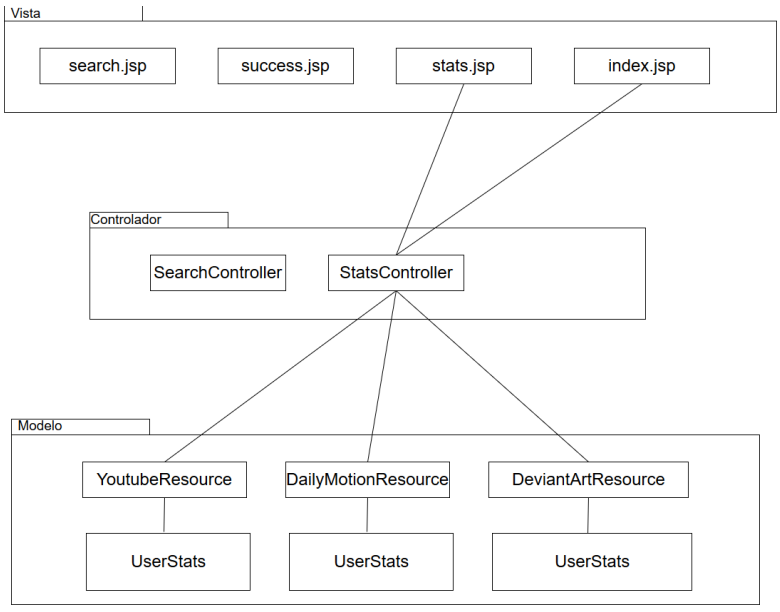



Diagrama MVC StatsImages



## 3.5 Diagramas de secuencia

Diagrama de secuencia MVC de búsqueda

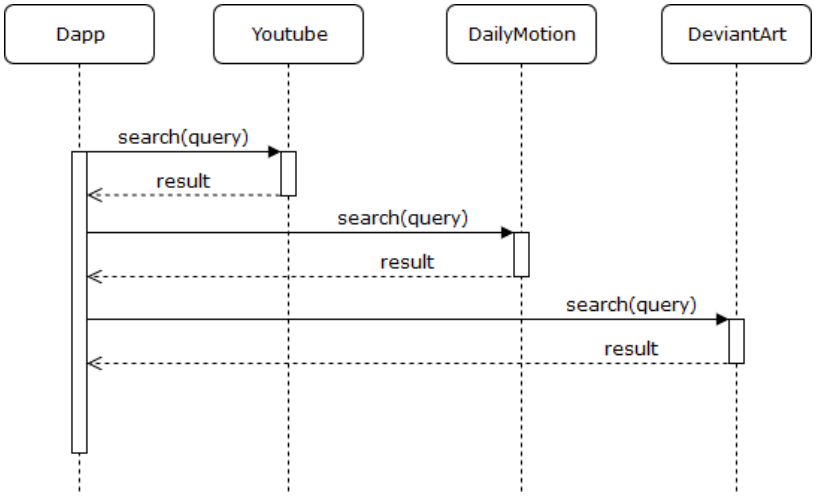
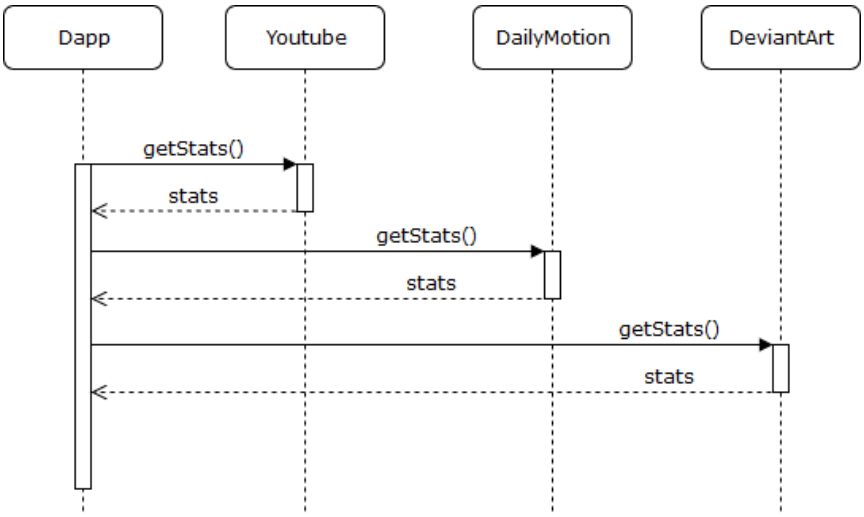



Diagrama de secuencia MVC de estadísticas de usuario




# 4 Implementación

Sin duda los métodos que más problemática nos han generado a la hora de implementarlos han sido los de publicación de comentarios tanto en YouTube como en DeviantArt y ambos se implementan de forma un poco diferente a pesar de tratarse del mismo método HTTP.

Comenzaremos con el de YouTube:

El método recibe como parámetros la URL a la cual tendrá que ejecutar el método HTTP y la ID tanto del canal como del vídeo donde se comentará. Dentro del método declaramos la variable *text* que recogerá la información del input donde escribimos el comentario para luego ponerlo en el cuerpo del método. En esta parte del método podemos observar la diferencia entre este y el de DeviantArt, a la hora de construir el cuerpo (ya que el de YouTube requiere de más información) y mandar la petición (ya que el método asíncrono fetch que se usa con DeviantArt y el resto de POSTs no permitía construir un cuerpo en formato JSON, daba un error a la hora de parsear).

YouTube recibe y devuelve en formato JSON, luego tenemos que indicárselo en la cabecera. Por último, si la operación es llevada a cabo con éxito, se nos devuelve la respuesta del método en formato JSON.

```
function postCommentYoutube(url1, videoId, channelId) {
    const chan = channelId;
    const vid = videoId;
    const text = document.getElementById(videoId + "-c").value;
    document.getElementById(videoId + "-c").value = "¡Comentario publicado con éxito!";
    console.log("Channel" + channelId + " ::::: " + "Video " + videoId);
    console.log(url1);
    const URL = url1;

    (async () => {
          const rawResponse = await fetch(URL, {
            method: 'POST',
            headers: {
              'Accept': 'application/json',
              'Content-Type': 'application/json'
            },
            body: JSON.stringify({snippet: {channelId: chan, videoId: vid, topLevelComment: {snippet: {textOriginal: texto}}}})
          });
          const content = await rawResponse.json();

          console.log(content);
        })();
}
```

Para el método de DeviantArt recibimos los mismos parámetros y en la variable Data formamos el cuerpo del método recogiendo solamente el comentario que se ha escrito. En este método, el contenido no está escrito en JSON, así que lo indicamos en la cabecera. Por último, ejecutamos el método pidiendo que nos den la respuesta en formato JSON a modo de confirmación.

```
function postComentarioDA(url1, token1, id1) {
    const access_token1 = token1;
    const devid = id1;
    console.log(token1 + " ::::: " + url1);
    const URL = url1 + "?access_token=" + access_token1;
    const Data = "body=" + document.getElementById(devid + "-c").value;
    document.getElementById(devid + "-c").value = "¡Comentario publicado con éxito!";
    console.log(Data);
    const othePram= {
        method: 'POST',
        //mode: "no-cors",
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
             },
        body: Data
       };

        //.then(response=>console.info(response.type))// opaque
    fetch(URL,othePram)
    .then(data=>{return data.json()})
    .then(res=>{console.log(res)})
    .catch(error=>console.log(error));
}
```

Por último, también nos gustaría destacar el método para sacar las estadísticas, ya que para YouTube tuvimos que implementarlo en un único método 'masivo' para no saturar la API de ésta con peticiones (tiene un límite de 10.000 puntos de peticiones). Así que implementamos un método desde el cual se pidieran todos los datos posibles:

```
public List<String> getStats() throws UnsupportedEncodingException {
    Integer comments = 0;
    Integer views = 0;
    Integer likes = 0;
    Integer dislikes = 0;
    String name = "";
    String id = "";
    String subscribers = "";

    List<GetUserVideosItem> items = getOwnYoutubeVideos().getItems();
    if (!items.isEmpty()) {
        VideoSnippet named = items.get(0).getSnippet();
        List<StatisticsItem> stats = getUserStatistics().getItems();

        comments = 0;
        views = 0;
        likes = 0;
        dislikes = 0;

        for (StatisticsItem st : stats) {
            comments += Integer.valueOf(st.getStatistics().getCommentCount());
            views += Integer.valueOf(st.getStatistics().getViewCount());

            likes += Integer.valueOf(st.getStatistics().getLikeCount());
            dislikes += Integer.valueOf(st.getStatistics().getDislikeCount());

        }

        name = named.getChannelTitle();
        id = named.getChannelId();

        // Subscribers
        String uri2 = "https://www.googleapis.com/youtube/v3/channels?part=statistics&id=" + id + "&access_token="
                + access_token;

        log.log(Level.FINE, "Youtube ChannelStats URI: " + uri2);

        ClientResource cr2 = new ClientResource(uri2);

        ChannelStats channelStats = cr2.get(ChannelStats.class);

        subscribers = channelStats.getItems().get(0).getStatistics().getSubscriberCount();

    }

    List<String> result = new ArrayList<>();
    result.add(name);
    result.add(id);
    result.add(String.valueOf(likes));
    result.add(String.valueOf(dislikes));
    result.add(String.valueOf(comments));
    result.add(String.valueOf(views));
    result.add(subscribers);

    return result;
}
```

En el método recogemos datos de diferentes recursos de YouTube y devolvemos los que usamos en una lista de String para mayor facilidad a la hora de recuperarlos.
Por la misma razón, el método utilizado para sacar el mejor vídeo propio de YouTube se realiza en el propio StatsController, para así evitar llamadas reiteradas a la API.

```
// Sacando mejor vídeo
Integer viewsBestVideo = 0, resultViews = 0;
List<StatisticsItem> items = youtubeStatistics.getItems();

if (!items.isEmpty()) {
    StatisticsItem youtubeMostViewed = youtubeStatistics.getItems().get(0);
    for (StatisticsItem mv : youtubeStatistics.getItems()) {
        viewsBestVideo = Integer.valueOf(mv.getStatistics().getViewCount());
        resultViews = Integer.valueOf(youtubeMostViewed.getStatistics().getViewCount());

        if (viewsBestVideo > resultViews) {
            youtubeMostViewed = mv;
        }
    }

    String youtubeMostViewedTitle = "";
    for (GetUserVideosItem gi : youtubeVideos.getItems()) {
        if (youtubeMostViewed.getId().equals(gi.getId().getVideoId())) {
            youtubeMostViewedTitle = gi.getSnippet().getTitle();
        }
    }

    req.setAttribute("youtubeMostViewed", youtubeMostViewed);
    req.setAttribute("youtubeMostViewedTitle", youtubeMostViewedTitle);
}
```

# 5 Pruebas

| Resumen | |
|---|---|
| **Número total de pruebas realizadas** | 12 |
| **Número de pruebas automatizadas** | 0 (0%) |

| **ID** | **Prueba 1** |
|---|---|
| **Descripción** | Prueba para la detección de errores al implementar búsquedas en YouTube. |
| **Entrada** | Se hace uso de la API YouTubeResource usando la URL /SearchController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| **ID** | **Prueba 2** |
|---|---|
| **Descripción** | Prueba para la detección de errores al implementar búsquedas en Dailymotion. |
| **Entrada** | Se hace uso de la API DailymotionResource usando la URL /SearchController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| **ID** | **Prueba 3** |
|---|---|
| **Descripción** | Prueba para la detección de errores al implementar búsquedas en DeviantArt. |
| **Entrada** | Se hace uso de la API DeviantArtResource usando la URL /SearchController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 4** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver la lista de favoritos en Dailymotion. |
| **Entrada** | Se hace uso de la API DailymotionResource usando la URL /SearchController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 5** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver la lista de likes en YouTube. |
| **Entrada** | Se hace uso de la API YoutubeResource usando la URL /SearchController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 6** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver la lista de favoritos en DeviantArt. |
| **Entrada** | Se hace uso de la API DeviantArtResource usando la URL /SearchController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 7** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver la lista de “watchlater” en Dailymotion. |
| **Entrada** | Se hace uso de la API DailymotionResource usando la URL /SearchController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 8** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver las stats de YouTube. |
| **Entrada** | Se hace uso de la API YoutubeResource usando la URL /StatsController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 9** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver las stats de Dailymotion. |
| **Entrada** | Se hace uso de la API DailymotionResource usando la URL /StatsController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 8** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver las stats de DeviantArt. |
| **Entrada** | Se hace uso de la API DeviantArtResource usando la URL /StatsController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 9** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver la lista de dislikes en YouTube. |
| **Entrada** | Se hace uso de la API YoutubeResource usando la URL /SearchController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 10** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver la mejor publicación de DeviantArt. |
| **Entrada** | Se hace uso de la API DeviantArtResource usando la URL /StatsController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

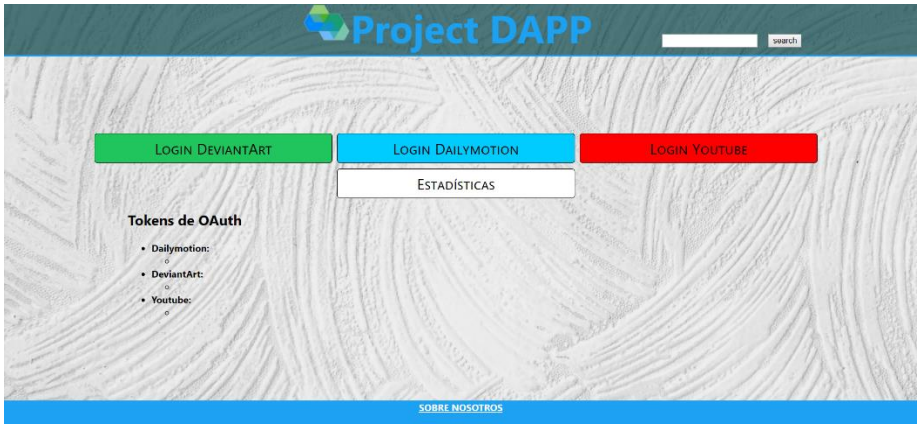
| ID | **Prueba 11** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver la mejor publicación en YouTube. |
| **Entrada** | Se hace uso de la API YoutubeResource usando la URL /StatsController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

| ID | **Prueba 12** |
|---|---|
| **Descripción** | Prueba para la detección de errores al devolver la mejor publicación de Dailymotion. |
| **Entrada** | Se hace uso de la API DailymotionResource usando la URL /StatsController. |
| **Salida esperada** | Los datos devueltos en formato JSON son mapeados a una clase Java y a continuación se muestran por pantalla. |
| **Resultado** | **ÉXITO** |
| **Automatizada** | No |

# 6 Manual de usuario

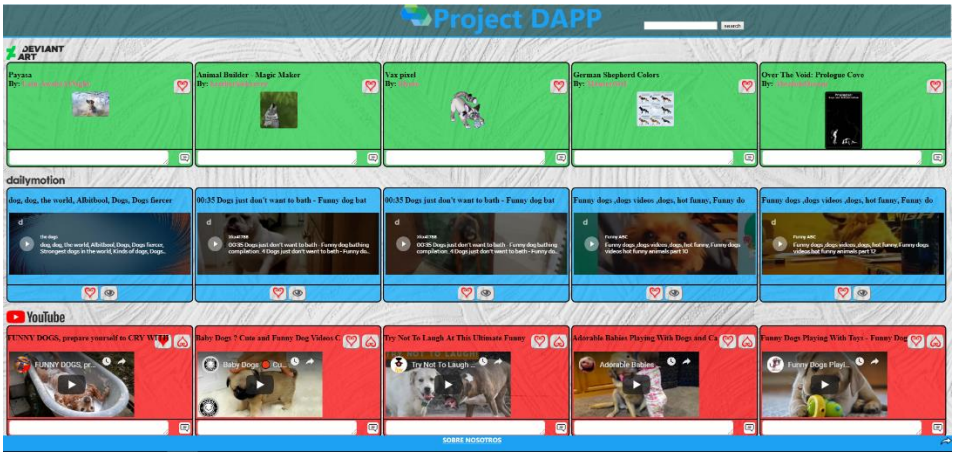
## 6.1 Mashup

La página principal de la aplicación actualmente es la siguiente:




Si entramos en cualquiera de los botones de inicio de sesión, entraremos en la aplicación otorgándole los derechos especificados en el scope.

Si hacemos la búsqueda con el tema especificado, la página será la siguiente:




Si entramos en el botón de estadísticas pasaremos a la siguiente vista:


Por último, la vista de la página de información sobre el proyecto:

## 6.2 API REST

Todos las URI comienzan de la siguiente manera: /api/m/ y los recursos serán devueltos en formato JSON.

| | | |
|---|---|---|
| **GET** | /all | Devuelve todas las imágenes y vídeos. |
| **GET** | /all//query/{query} | Devuelve las imágenes y vídeos que contengan la query en el título. |
| **GET** | /images | Devuelve todas las imágenes. |
| **GET** | /images/query/{query} | Devuelve las imágenes que contengan la query en el título. |
| **GET** | /images/author/{author} | Devuelve las imágenes cuyo autor presente coincidencias con author. |
| **GET** | /images/page/{page} | Devuelve la lista de imágenes paginada por la página page. |
| **GET** | /images/id/{id} | Devuelve una imagen con la id especificada. |
| **GET** | /videos | Devuelve todos los vídeos. |
| **GET** | /videos/query/{query} | Devuelve los vídeos que contengan la query en el título. |
| **GET** | /videos/author/{author} | Devuelve los vídeos cuyo autor presente coincidencias con author. |
| **GET** | /videos/page/{page} | Devuelve la lista de vídeos paginada por la página page. |
| **GET** | /videos/id/{videoId} | Devuelve un video con una id determinada. |
| **POST** | /addImage | Añade una imagen. |
| **POST** | /addVideo | Añade un vídeo. |
| **PUT** | /updateImage | Actualiza una imagen antigua. |
| **PUT** | /updateVideo | Actualiza un vídeo antiguo. |
| **DELETE** | /deleteImage/{imageId} | Elimina la imagen con la id especificada. |
| **DELETE** | /deleteVideo/{videoId} | Elimina el vídeo con la id especificada. |

Para los métodos POST y PUT deberemos hacer la petición con un cuerpo en formato JSON con los siguientes parámetros:

| | |
|---|---|
| **id** | Parámetro tipo String que identifica a un objeto. |
| **autor** | Parámetro tipo String indicando el autor del recurso. |
| **titulo** | Parámetro tipo String indicando el título del recurso. |

Códigos de estado del servicio:

| | | |
|---|---|---|
| **200** | **OK** | La solicitud ha sido procesada correctamente. |
| **201** | **CREATED** | El recurso se ha creado con éxito. |
| **204** | **NO CONTENT** | El recurso se ha actualizado o eliminado con éxito. |
| **400** | **BAD REQUEST** | La petición es incorrecta. |
| **401** | **UNAUTHORIZED** | La petición requiere una autenticación. |
| **404** | **NOT FOUND** | No fue posible encontrar los recursos requeridos. |
| **500** | **INTERNAL SERVER ERROR** | Error inesperado en el servidor. |

# Referencias

[1] *Balsamiq*. http://balsamiq.com/. Accedido en Enero 2014.

[2] J. Webber, S. Parastatidis y I. Robinson. *REST in Practice: Hypermedia and Systems Architecture.* O'Reilly Media. 2010.