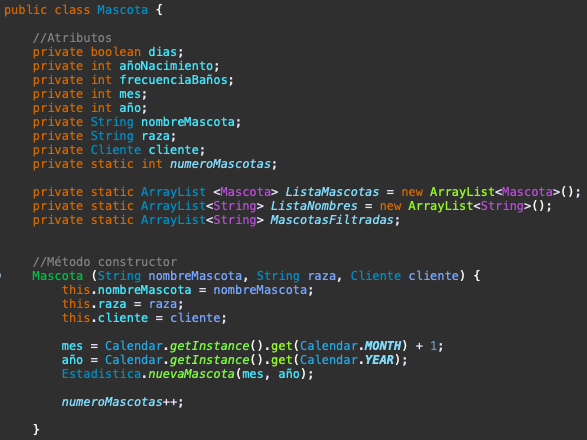
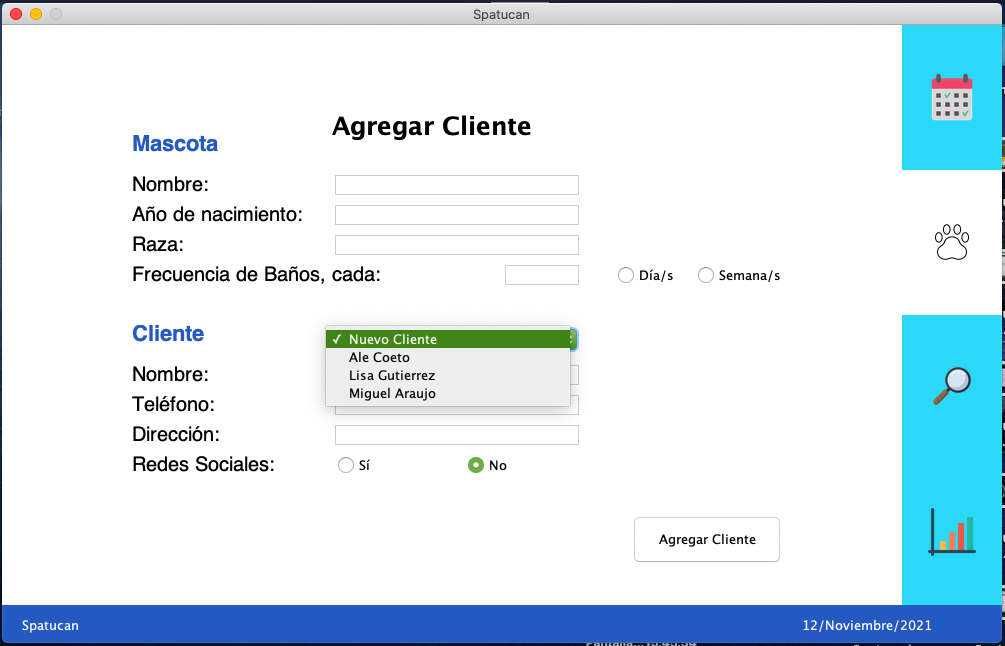
**Criterio C: Desarrollo**

1. **Programación orientada a objetos**

Para la solución informática se utilizó el modelo de POO debido a que esta técnica permite representar objetos y establecer distintos tipos de relaciones necesarias para cumplir con los primeros tres criterios de logro: poder crear, modificar y eliminar clientes o citas (de los diferentes tipos) para mantener un registro ordenado, además de hacer estadísticas para cada mes. Por lo tanto, se crearon los siguientes objetos:

**Mascota:** usa atributos para representar una mascota y a través de la agregación, se le asigna un cliente, ya que este podría existir por sí solo.



****

**Cliente:** permite representar a un dueño quien podría tener más de una mascota, por lo que al crear una nueva es posible buscar un cliente ya existente y reutilizar su código. De igual manera, la agregación es útil al editar información de un cliente, ya que solo se tiene que modificar ese objeto una vez.

|  |  |
| --- | --- |

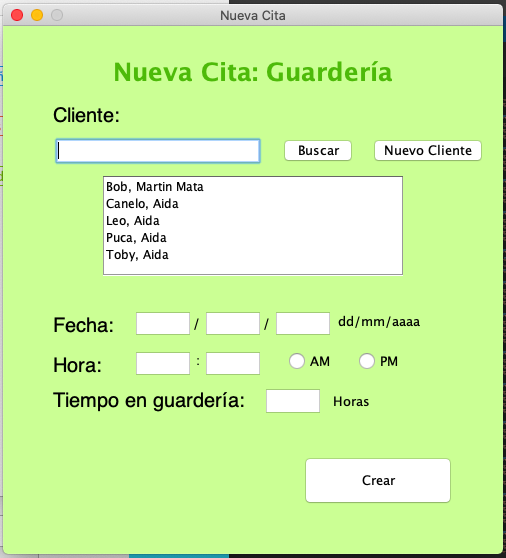
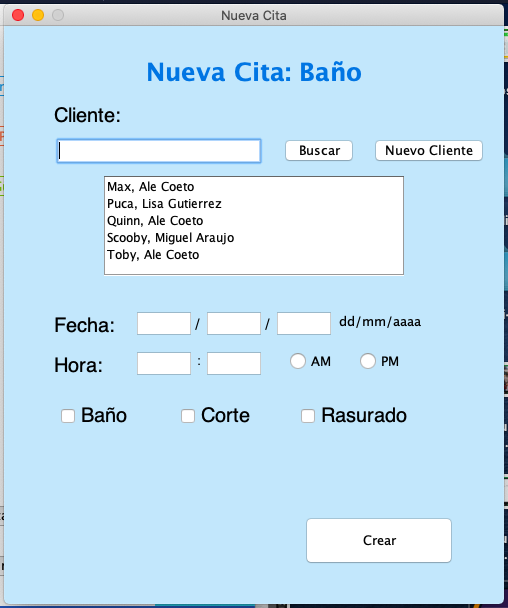
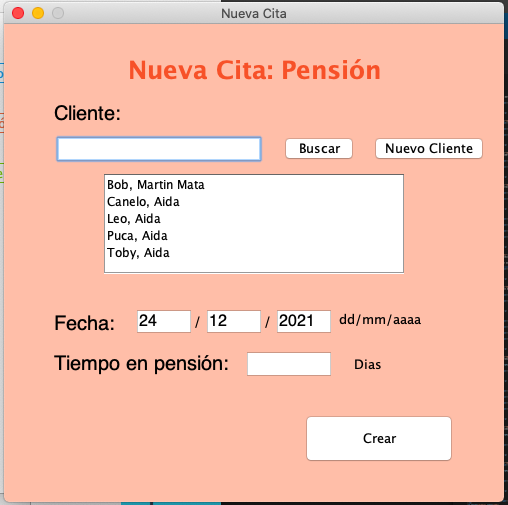
La POO también fue implementada para usar los beneficios de un modelo de herencia y reutilizar código al tener varios tipos de citas:

**Cita:** superclase que tiene atributos necesarios para agendar una cita de cualquier tipo y también se usa la agregación para asignarle un cliente.



**Baño, Pension y Guarderia:** heredan de la superclase Cita, pues cada uno tiene atributos y métodos propios (algunos son reescritos) y además, cada uno tiene un método constructor para diferentes datos de entrada.





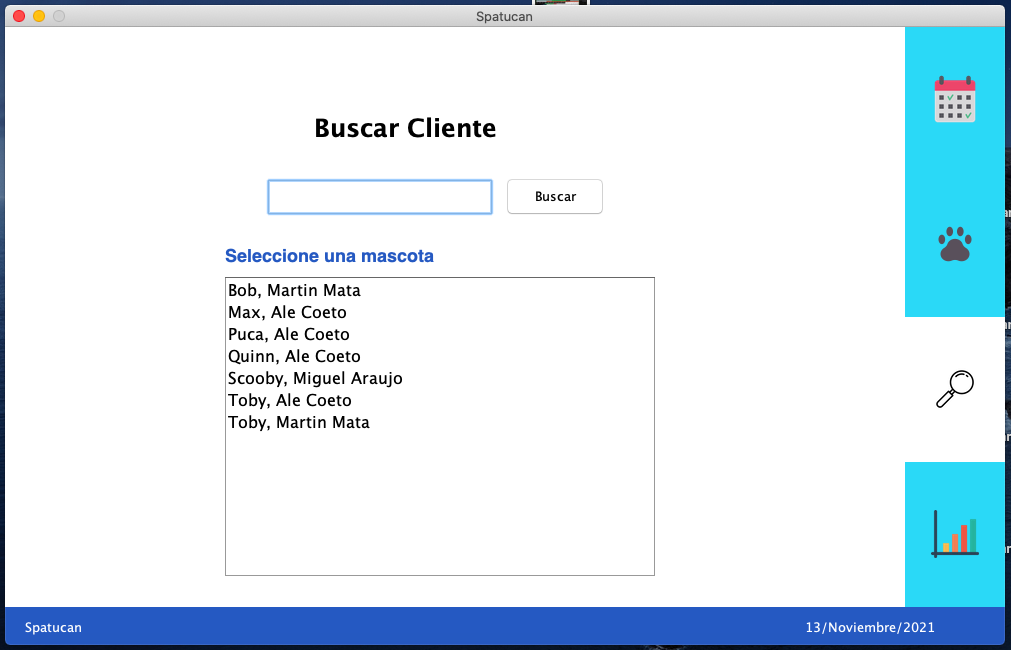
Además, gracias al polimorfismo, varios métodos fueron sobrescritos para adaptarse a cada tipo de cita en lugar de tener que crear métodos completamente nuevos para cada una.



1. **Algoritmos de ordenamiento**

Las dos listas del objeto Mascota (una con los objetos Mascota y una con el nombre de la mascota y el cliente) son ordenadas alfabéticamente para mantener un orden y facilitar el acceso a las mascotas en la interfaz. Al ordenar las listas, cada vez que se agrega una nueva, solo es necesario ordenar el último elemento de cada lista. Por lo tanto, se usa un algoritmo de ordenamiento similar a un bubble sort, pero con una notación en el peor caso de O(), pues iniciando con el elemento al final de la primera lista se compara si es menor al de su izquierda, y en caso de serlo se intercambian tanto los elementos de esa lista como los de la segunda lista, por lo que si no hay intercambio, las listas ya están en orden y se termina el algoritmo.



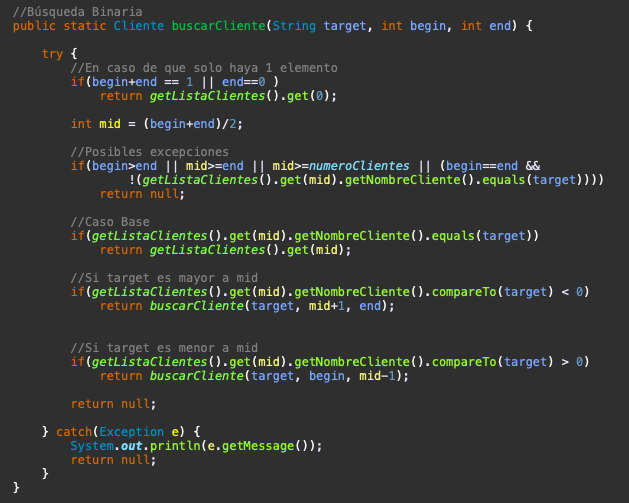


Para ordenar los objetos Cliente alfabéticamente, se utiliza el mismo algoritmo pero simplificado al ser solo una lista, lo que también facilita la búsqueda de un cliente.

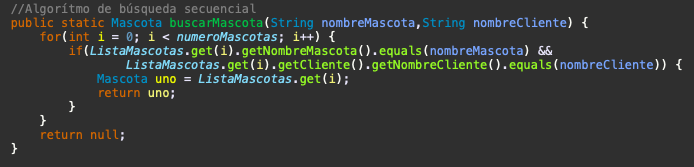
|  |  |
| --- | --- |

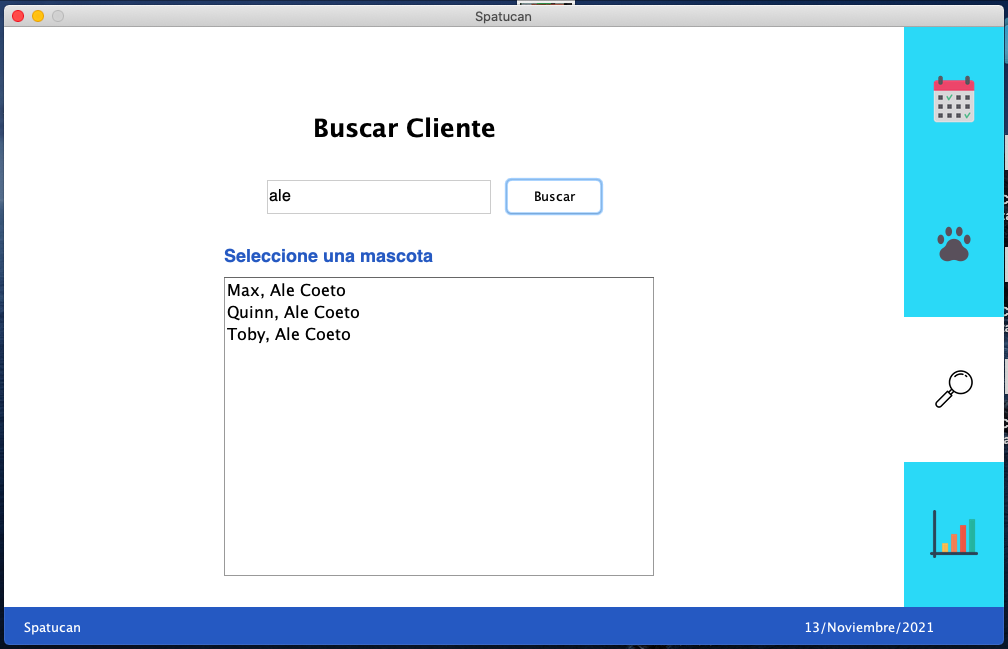
1. **Algoritmos de búsqueda binaria y secuencial**

Para cumplir con el primer criterio de logro y poder acceder a la información, modificar datos o eliminar clientes o mascotas, en el programa se utilizan dos tipos de algoritmos de búsqueda. El primero es un algoritmo recursivo de búsqueda binaria que fue usado debido a su alta eficiencia con notación O() para buscar clientes, ya que la lista se encuentra en orden alfabético y se espera que crezca al paso del tiempo, por lo que esta técnica ayuda a reducir los tiempos de búsqueda a pesar de que más elementos sean agregados.

****

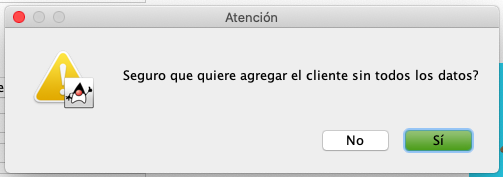
Por otro lado, para buscar mascotas, es más adecuado una búsqueda secuencial con notación O() debido a que es posible que al tratarse de mascotas, existan varias con el mismo nombre, por lo que se tiene que comparar tanto el nombre de la mascota como el del cliente, lo cual no sería eficiente con una búsqueda binaria, pero sí con una secuencial.

****

****

1. **Validación**

Varias validaciones son realizadas para evitar accidentes al confirmar que el cliente desea hacer una acción cómo el crear una mascota a pesar de no haber ingresado todos los datos o el eliminar un cliente o cita.



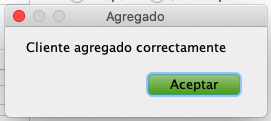
Para evitar errores se verifican todas las entradas de los cuadros de textos u otros botones a través del manejo de excepciones y en caso de haber una, ésta es mostrada al cliente para ser corregida.





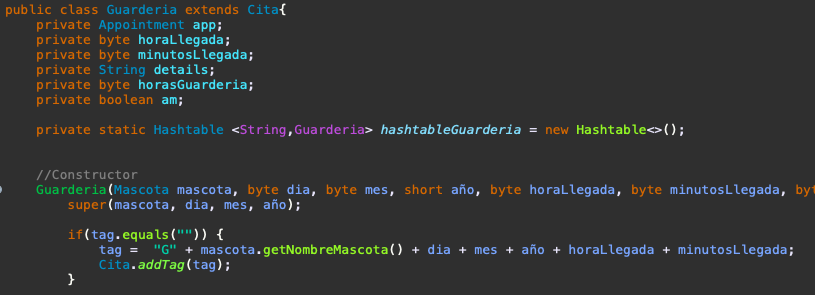


Adicionalmente, al realizar algunas acciones, se muestran pop-ups para que el cliente reciba retroalimentación de la aplicación y sepa que algo funcionó adecuadamente o no, ayudando a cumplir con el último criterio de logro: que la interfaz sea fácil de usar.



1. **Pensamiento computacional**

Para poder adaptar la librería JPlanner al programa y mostrar las citas en un calendario, lo cual es parte del último criterio de logro, fue necesario implementar un hashtable en cada tipo de cita, el cual contenía el objeto del tipo de cita y un *tag* para identificarlo, y además, en la superclase Cita se tuvo que crear un ArrayList con todos los tags. Esto se debe a que la librería mantiene una lista con los objetos del calendario en el orden en que fueron agregados, por lo que para eliminar uno, es necesario saber el índice del objeto, el cual sería el mismo que el de la lista de tags. De igual manera, este proceso simplifica el mantenimiento del programa al conectar los objetos de la librería con los del programa por medio del tag.

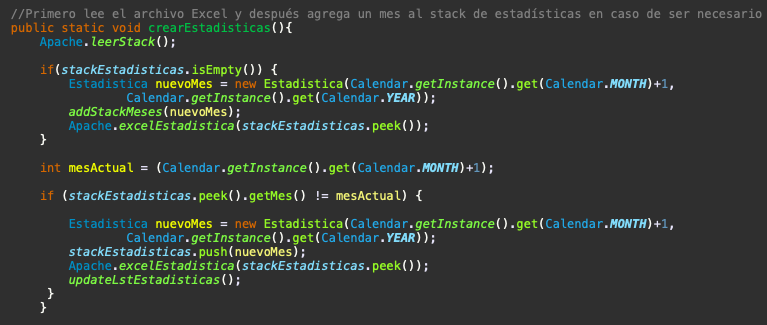


Adicionalmente, para identificar una cita seleccionada, se obtuvo el tag del objeto elegido en el calendario y se identificó el tipo de cita que era de acuerdo a su tag, y usando el hashtable, fue posible acceder a la cita y editarla o borrarla.



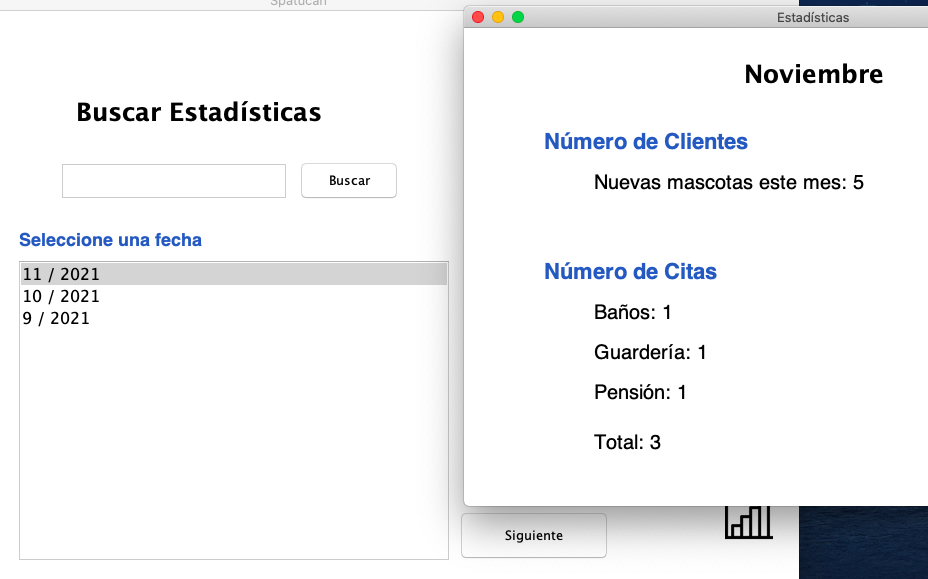


Por otro lado, las estadísticas debían hacerse automáticamente para cada mes. Para ello, se creó un algoritmo para agregar un nuevo mes en caso de ser necesario y cumplir con uno de los criterios de logro.



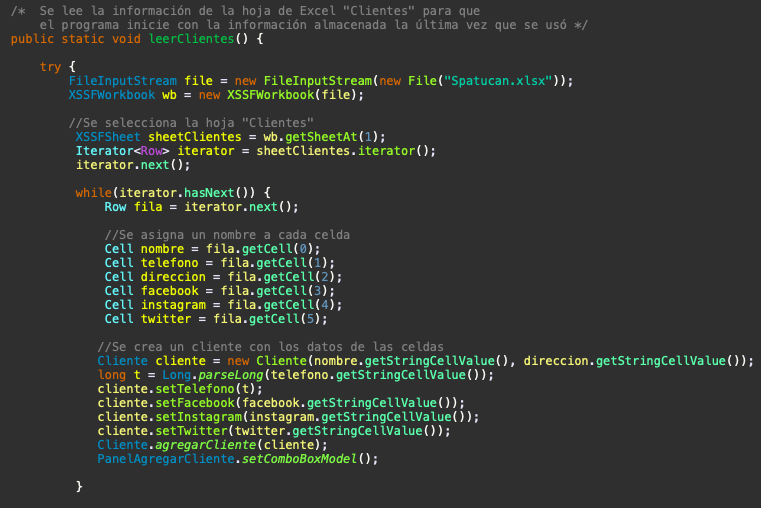
De igual manera, gracias al uso de una pila, fue posible acceder rápidamente al último mes creado y así actualizar la información en caso de ser necesario.

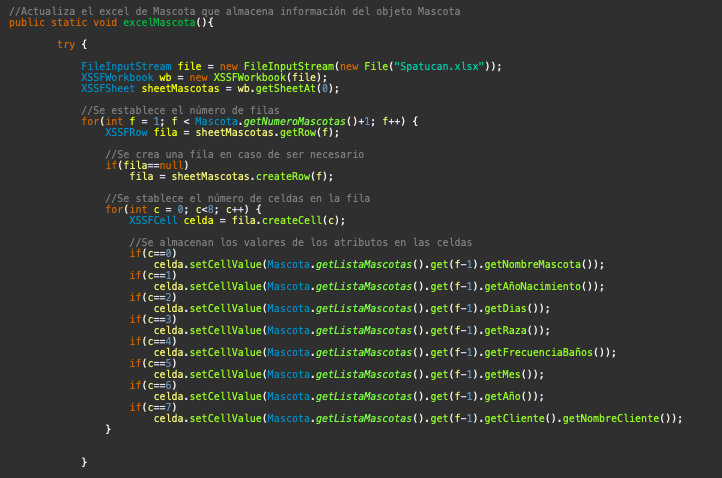




1. **Almacenamiento**

Para almacenar la información (necesario para todos los criterios de logro), se implementó la librería Apache Poi para utilizar archivos de Excel, pues estos permitieron guardar la información de cada uno de los objetos en diferentes hojas del mismo archivo, lo cual también simplificó el proceso de leer y cargar la información al abrir nuevamente la aplicación. Además, al almacenar los datos en filas y columnas, se mantiene un órden que facilita el mantenimiento del programa, pues en caso de ser necesario, el cliente o asesor pueden acceder directamente al archivo para hacer modificaciones. Inclusive, el cliente podría ver y entender la información de todas las mascotas y citas agendadas a través del archivo xlsx.





952 palabras

**Bibliografía**

Apache (2021). *Apache POI - the Java API for Microsoft Documents.* Recuperado el 04/10/2021 de: <https://poi.apache.org/>

Bro Code (2020). *Java GUI*. Recuperado el 22/07/2021 de: <https://www.youtube.com/watch?v=Kmgo00avvEw&t=10461s>

Códigos de Programación. (2017). *10.* *Crear, Leer y Modificar Excel en Java y MySQL.* Recuperado el 06/10/2021 de: <https://www.youtube.com/watch?v=oG18JTIKtLo>

Fun2Learn (2019). *How to read excel file in Java?*. Recuperado el 06/10/2021 de: <https://www.youtube.com/watch?v=Mrtw7NvMQvw&t=249s>

Mindfusion (2020). *JPlanner Online Documentation.* Recuperado el 02/08/2021 de: <https://www.mindfusion.eu/onlinehelp/jplanner/index.htm>

Oracle (2020). *Class JList<E>*. Recuperado el 30/07/2021 de: <https://docs.oracle.com/javase/7/docs/api/javax/swing/JList.html>

Oracle (2020). *Package javax.swing.* Recuperado el 22/07/2021 de: <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

PildorasInformaticas (2017). *Curso Java. Anexo. Swing Av I JList. Vídeo 265*. Recuperado el 30/07/2021 de: <https://www.youtube.com/watch?v=9xuZrLsTeFM>