ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

Aplicación de comunicación para personas con dificultades en el habla

Arantxa Mateo Bayo

Tutor: José Javier Astrain Escola

Pamplona, 17 de Julio de 2013

ÍNDICE

Introducción

1. INTRODUCCIÓN

1.1. PROBLEMA A RESOLVER

La idea del proyecto surge a partir de las necesidades de un caso concreto, y con el deseo de ofrecer ayuda a personas con discapacidad psíquica o con graves afectaciones en el lenguaje.

La aplicación está dirigida a personas con una discapacidad física y/o psíquica que les impide, total o parcialmente, comunicarse con otras personas de su entorno. También podría facilitar la comunicación a personas con dificultades auditivas, o con algún tipo de problema que les dificulte de alguna forma la comunicación.

Consideramos que esta aplicación va a ofrecer ciertas ventajas sobre otros sistemas alternativos de comunicación:

-Permitir el acceso a los pictogramas de una forma rápida y sencilla.

-Las imágenes pictográficas se asemejan mucho a las imágenes reales y fácilmente reconocibles.

-Sencillez de compresión por parte de personas ajenas al entorno educativo, lo que facilita la comunicación con sujetos no familiarizados con este tipo de sistemas.

-Facilita la formación de frases sencillas al seleccionar los símbolos de una forma muy intuitiva.

El objetivo es encontrar una forma fácil e intuitiva de comunicación, que les permita adaptarse más fácilmente a su entorno, evitando barreras que les pueda causar estas dificultades.

Es una aplicación creada para que no se necesite un aprendizaje previo de su funcionamiento sino que sea intuitivo y que se adapte a posibles problemas que se puedan tener en el aprendizaje, y les permitirá además a aquellas personas con dificultades intelectuales adquirir nuevos conocimientos de forma sencilla.

Teniendo en cuenta que los usuarios podrían llegar a tener incluso algún tipo de dificultad visual, por lo que es necesario que se utilicen imágenes que se vean con claridad y sin demasiado detalle.

Por tanto, el problema a resolver principalmente permitir a las personas que tengan cualquier tipo de problema que les impida comunicarse la posibilidad de hacerlo de una forma sencilla para ellos y entendible para las personas que los rodean.

1.2. OBJETIVO

Los objetivos a alcanzar con esta aplicación son los que se pueden ver a continuación:

- Permitir reconocer e identificar las palabras mediante las imágenes.
- Asociar las palabras a la imagen para una utilización más sencilla.
- Asociación de los sonidos a las palabras correspondientes.
- Facilitar la comunicación con las personas que los rodean.
- Ver cómo evoluciona el tiempo.
- Adaptarse al paso del tiempo a la hora de realizar una actividad.

Todos estos objetivos se pretenden realizar mediante una aplicación interactiva en la que se permita extrae el máximo rendimiento que poseen este tipo de personas a la hora de realizar un aprendizaje visual.

Dado que es una aplicación destinada a persona con discapacidad intelectual, se tendrá en cuenta que los elementos que la formen tendrán que ser sencillos y de fácil identificación por parte de ellos.

También hay que hacer una adecuada elección de los objetos a representar dentro de la aplicación, teniendo en cuenta su fácil identificación por parte del usuario y que representen con claridad la realidad sin buscar que sean fotos con demasiado detalle, haciendo que estas se asemejen a las que las personas con deficiencia intelectual utilizan para comunicarse cuando utilizan pictogramas.

1.3. ESTADO DEL ARTE

La plataforma elegida para el desarrollo ha sido el sistema operativo IOS de Apple para dispositivos móviles y tabletas, ya que lo que queremos es permitir que las personas puedan acceder a la aplicación de una forma sencilla.

Apple incorpora tecnología asistencial en sus productos sin coste adicional. Por ejemplo, iPhone, iPad, iPod y OS X incluyen ampliación de pantalla y VoiceOver, un lector de pantalla para invidentes. También incluyen una interfaz de usuario alternativa y simplificada que favorece la exploración y el aprendizaje, especialmente útil para personas con discapacidad cognitiva y de aprendizaje.

Los dispositivos de Apple incorporan como se puede observar numerosas ayudas para favorecer el aprendizaje de todo tipo de personas que tengan algún tipo de discapacidad.

Este tipo de plataformas, como es el iPad permiten al usuario interactuar, lo que hace que modifique la forma de trabajo, obteniendo resultados más inmediatos y motivadores, para las personas con discapacidad, en especial para los niños. Además les permite hacerlo de una forma más sencilla e intuitiva para ellos, ya que son dispositivos táctiles.

El iPad puede fácilmente convertirse en una herramienta de comunicación de alto rendimiento, dándole una voz propia a las personas que hasta hace poco debían conformarse con tarjetas visuales o con artefactos poco sofisticados que limitaban el desarrollo de sus habilidades.

En el mercado podemos encontrar otras aplicaciones creadas para facilitar a la personas con discapacidad intelectual la comunicación. A continuación especifico algunas de ellas.

- *SC@UT*

Es una herramienta que ayuda a la comunicación y al aprendizaje de las personas con necesidades especiales.

Esta herramienta creada por la Universidad de Granada, esta disponible para dispositivos Windows, Linux y Nintendo DS.

Aunque me parece una plataforma útil, las plataformas táctiles que hay hoy en día en el mercado son más sencillas de utilizar e intuitivas, comparadas con la dificultad que puede crear la utilización de un ordenador o de un dispositivo del tipo de la Nintendo DS, que necesitan del manejo del ratón y lápiz respectivamente.

Las nuevos dispositivos, tabletas, móviles... nos permiten una interacción más sencilla que hace que la persona vea resultados instantáneos sin necesidad de tener un conocimiento previo del funcionamiento del dispositivo que esta utilizando.

Este dispositivo en cambio puede ser útil para aumentar el nivel de dificultad llegado un momento y ayudar a que sepan utilizar el ordenador, ya que es gratuito.

- *Hotuba*

Esta aplicación permite la comunicación mediante pictogramas asociados a sonidos y su intención es también ayudar en la comunicación de las personas que tienen problemas en el habla.

Es un programa para tabletas de tipo Android, que reproduce el sonido de la tarjeta que se ha pulsado.

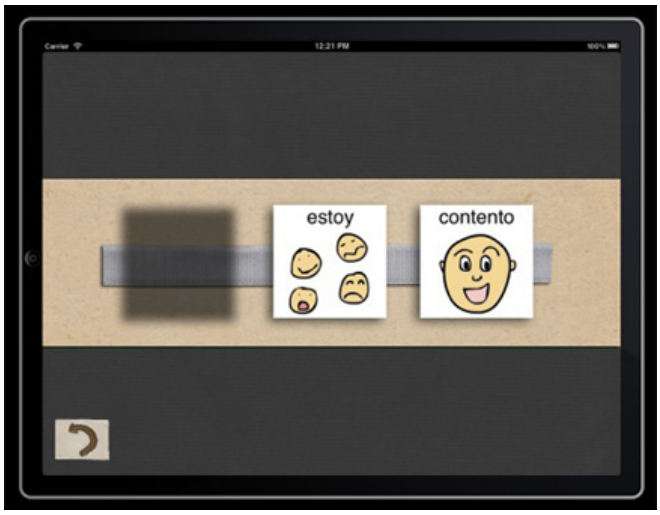
La principal diferencia es que esta aplicación permite únicamente la selección de una tarjeta, y no la creación de una frase, como es nuestro caso. Es el ejemplo más similar encontrado que utiliza el sistema Android para tabletas competencia principal de IOS.

- *DIME*

La aplicación más similar a la de este proyecto es DIME, aplicación para personas con discapacidad intelectual que permite también la elaboración de frases y la planificación del tiempo como en nuestro caso. Es para dispositivos iPad como ya he dicho es la que más se asemeja a este proyecto.

Se puede decir, que es una aplicación muy útil y posiblemente la más completa que permite realizar incluso cambios para ayudar a la compresión de los pictogramas.

El principal problema que podría tener esta aplicación sería el coste que tiene, ya que es bastante cara. Existe una versión gratuita pero que solo permite crear un tipo de frase de quiero/estoy y que no permite el acceso a crear más tipos de frases, a la modificación de tarjetas ni a la utilización del planificador de tiempo.



1.4. DESCRIPCIÓN DE LA PROPUESTA

Se propone realizar una aplicación IOS para dispositivos de tipo iPad, que ayude al usuario a comunicarse y a planificación el tiempo que puede dedicar una determinada actividad.

Por tanto, la aplicación contará de dos partes. La parte dedicada a la comunicación, que consistirá en la creación de frases y la reproducción de estas, y por otro lado, el planificador que hará una cuenta atrás del tiempo que le hemos asignado a una determinada actividad.

Como ya he comentado se necesitará de la emisión de sonidos. Tanto para la reproducción de las frases, como para avisar al usuario de la finalización del tiempo asignado a una actividad.

Para una mejor compresión de la aplicación se utilizarán pictogramas para representar las palabras y los componentes de las interfaces serán lo más simples posibles.

Para las fases en las que se divide la aplicación (análisis, diseño,...) se ha utilizado la metodología de Proceso Unificado UP que se caracteriza por ser iterativo e incremental y por estar dirigido por los casos de uso, además se ha utilizado UML para la elaboración de los diagramas.

Análisis

2. ANÁLISIS

En las primeras secciones de este Proyecto Fin de Carrera, se efectuará el análisis, diseño e implementación de la Aplicación para personas con dificultades en el habla. En la fase de análisis, se presentarán los requisitos y posteriormente se generará un análisis basado en diagramas de casos de uso.

2.1. ANÁLISIS DE REQUISITOS

El análisis de requisitos realiza un análisis detallado de la aplicación, qué es lo que la herramienta deberá hacer, y el modo en que debe hacerlo.

A continuación se muestra un listado de los requisitos que esta aplicación debe cumplir, separados en requisitos funcionales y no funcionales.

2.1.1. Análisis requisitos funcionales

A continuación aparecen los requisitos funcionales que debe cumplir la aplicación:

- La aplicación debe ayudar a la comunicación con otras personas.
- La herramienta debe tener imágenes que describan con claridad lo que representan.
- La aplicación debe reproducir las palabras seleccionadas por el usuario.
- Debe guiar al usuario a elegir las palabras en el orden correcto para formar una frase.
- La aplicación debe ser intuitiva, de fácil interacción.
- Debe ayudar a la persona a aprender sobre los sonidos de las palabras.
- Tendrá además animación para representar el paso del tiempo.
- Podrá ayudar al usuario a ser consciente del tiempo que transcurre.
- Ayudar a planificar el tiempo para la elaboración de una determinada actividad.
- Permitir volver a modificar la selección realizada.

2.1.2 Análisis requisitos no funcionales

Se describen, ahora los requisitos de carácter no funcional que podemos encontrar en la aplicación:

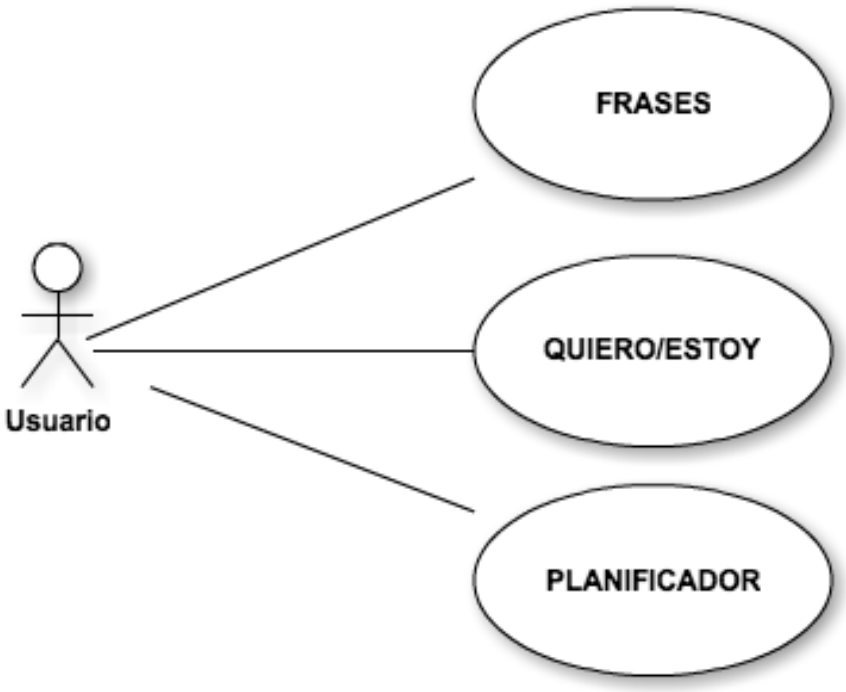
- Los gráficos que aparezcan debe ser acorde al usuario al que va dirigida, aunque lo pueda utilizar cualquier persona va dirigida a niños.
- Son imágenes sencillas de entender, y simples.
- Interfaz intuitiva y sin demasiadas cosas que distraigan la atención del usuario.
- Los elementos que componen la interfaz deben representar claramente su función.
- La voz que se reproduce debe ser clara a la hora de diferenciar las palabras.
- La aplicación debe funcionar en dispositivos de tipo iPad.

2.2. ANÁLISIS DE CASOS DE USO

A continuación se mostrarán los casos de uso correspondientes a la aplicación según los requisitos apartados, que se han enumerado anteriormente.

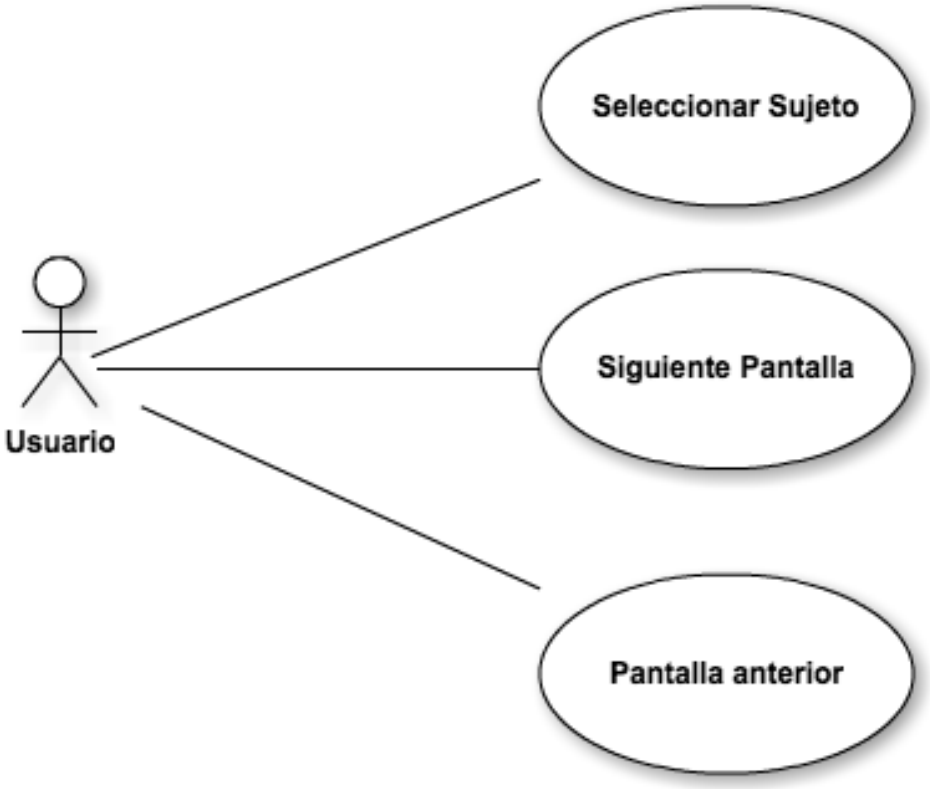
Para representar los casos de uso se mostrará un usuario y los distintos niveles, que representarán las partes de la aplicación.

2.2.1. CASO DE USO: PANTALLA PRINCIPAL



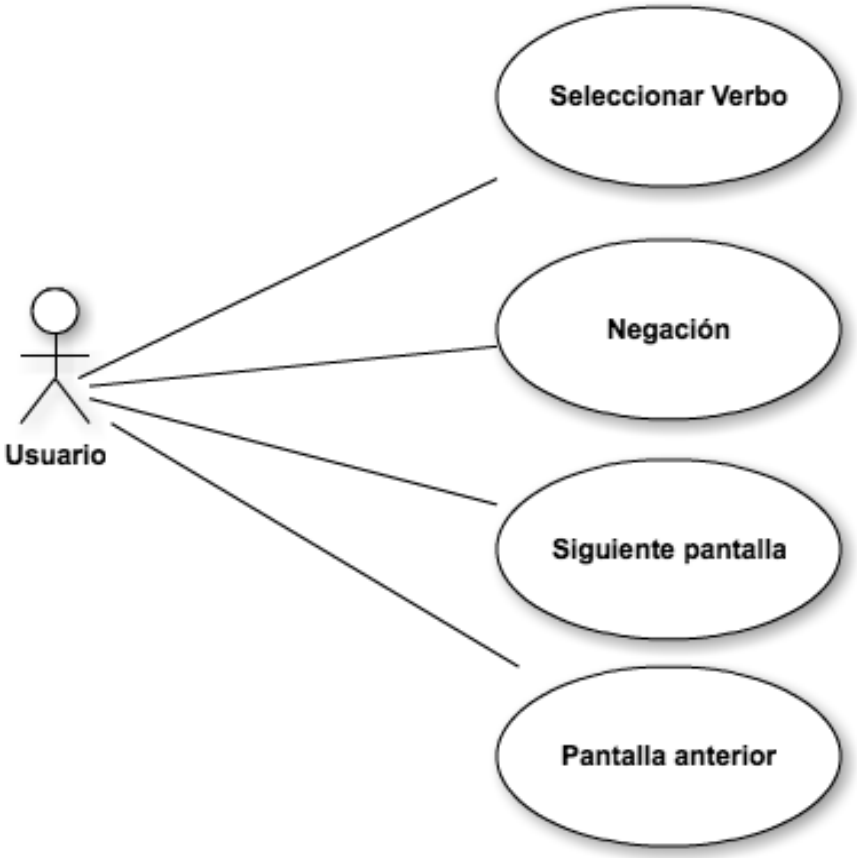
Descripción	Pantalla principal
Antecedentes	Ninguno
Opciones disponibles	• Crear frase completas: pasará a la pantalla de elección del sujeto. • Crear frase tipo Quiero/Estoy: pasará a la ventana para elegir quiero o estoy. • Planificador: irá a la ventana donde elegirá la actividad y el tiempo que se va a emplear.

2.2.2. CASO DE USO: PANTALLA SUJETO



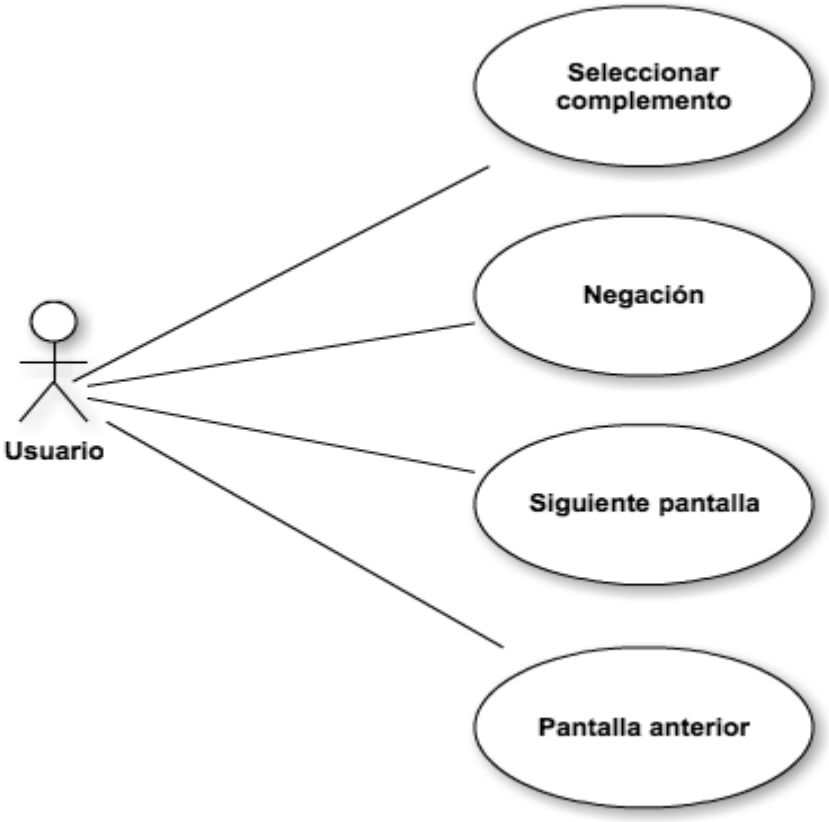
Descripción	Pantalla sujeto
Antecedentes	Haber seleccionado la opción de crear frases completas.
Camino básico	• Accede a la BBDD. • Muestra las opciones a seleccionar para sujeto. • El usuario selecciona un sujeto. • Guardamos la opción elegida, para las siguientes pantallas. • Selecciona ir a la siguiente pantalla.
Camino alternativo	• En caso de seleccionar volver a atrás (ir a pantalla principal). • Eliminar los datos en caso de que se hubiera seleccionado algo.

2.2.2. CASO DE USO: PANTALLA VERBO



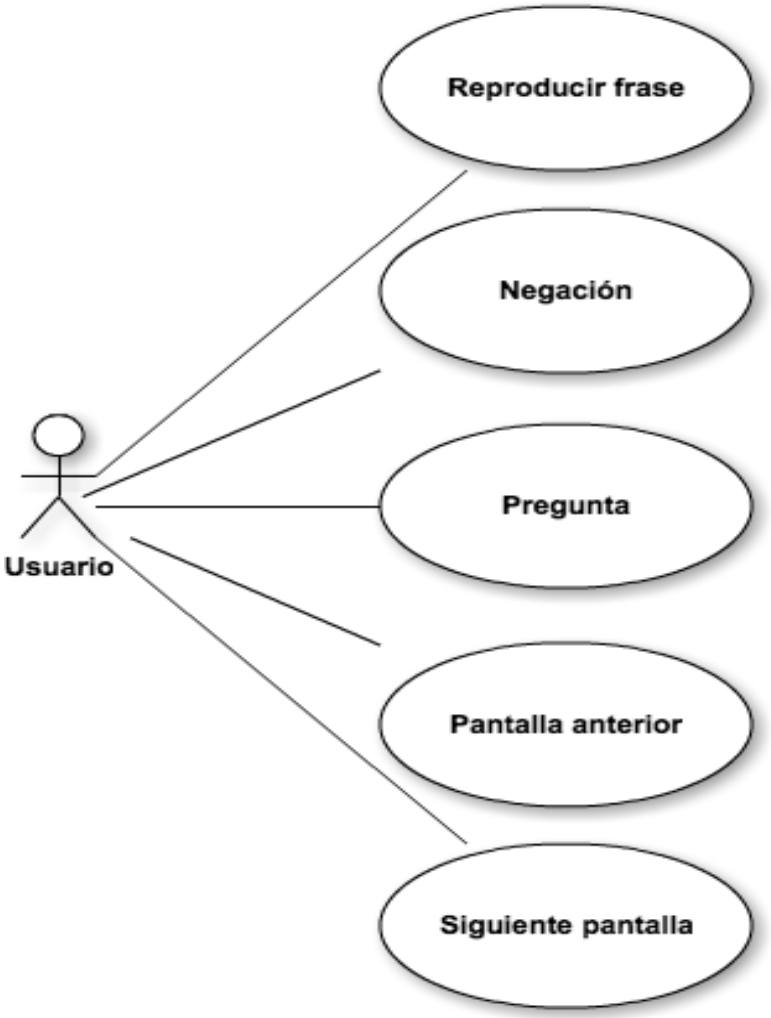
Descripción	Pantalla verbo
Antecedentes	Haber seleccionado un sujeto.
Camino básico	• Accede a la BBDD. • Muestra las opciones a seleccionar para verbo. • El usuario selecciona un verbo. • Guardamos la opción elegida, para las siguientes pantallas. • Selecciona ir a la siguiente pantalla.
Camino alternativo	• En caso de seleccionar volver a atrás (ir a pantalla principal), eliminar los datos en caso de que se hubiera seleccionado algo. • Negar la frase, en esta caso aparecerá una equis encima del verbo.

2.2.2. CASO DE USO: PANTALLA COMPLEMENTO



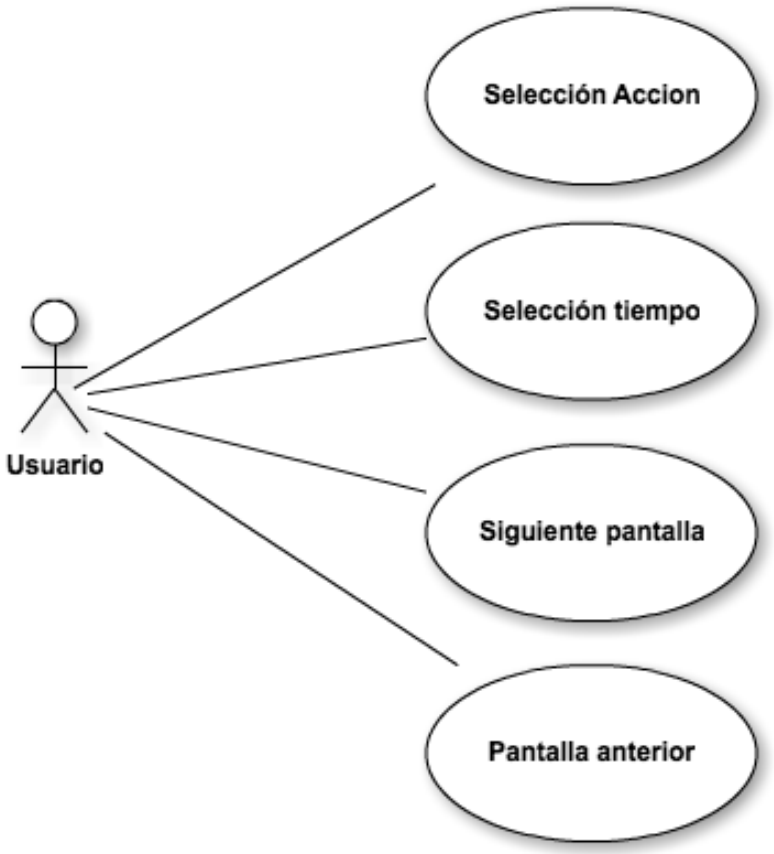
Descripción	Pantalla complemento
Antecedentes	Haber seleccionado un verbo.
Camino básico	• Accede a la BBDD. • Muestra las opciones a seleccionar para complemento. • El usuario selecciona un complemento. • Guardamos la opción elegida, para las siguientes pantallas. • Selecciona ir a la siguiente pantalla.
Camino alternativo	• En caso de seleccionar volver a atrás (ir a pantalla principal), eliminar los datos en caso de que se hubiera seleccionado algo. • Negar la frase, en esta caso aparecerá una equis encima del verbo.

2.2.2. CASO DE USO: PANTALLA FINAL



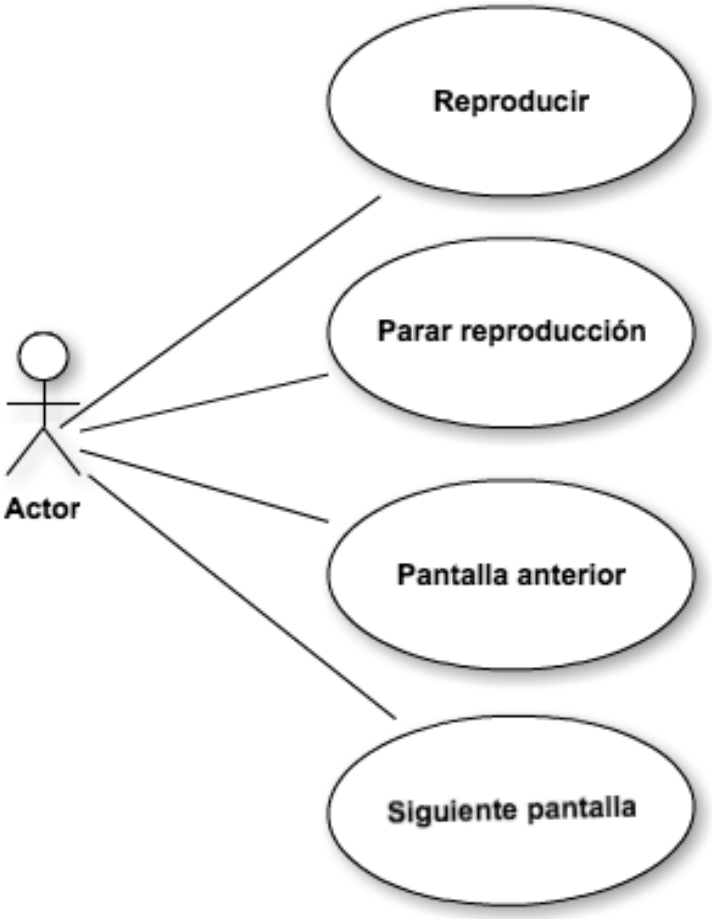
Descripción	Pantalla final
Antecedentes	Haber seleccionado algo en las pantallas de sujeto, verbo o complemento.
Opciones disponibles	• Reproducir la frase creada. • Negar la frase que se ha creado. • Hacer una pregunta. • Volver atrás. • Ir a la pantalla principal.

2.2.2. CASO DE USO: PANTALLA PLANIFICADOR



Descripción	Pantalla complemento
Antecedentes	Haber seleccionado la opción del planificador en la pantalla principal.
Camino básico	• Mostrar las acciones de la BBDD. • El usuario seleccionará una. • Seleccionará también el tiempo estimado. • Pasará a la pantalla de tiempo.
Camino alternativo	• Volver a la pantalla de inicio. • No seleccionar ninguna acción pero poner el cronometro, solo seleccionar tiempo.

2.2.2. CASO DE USO: PANTALLA TIEMPO



Descripción	Pantalla tiempo
Antecedentes	Haber seleccionado un tiempo en la pantalla del planificador
Opciones disponibles	• Reproducir la animación. • Parar la animación. • Ir a la pantalla anterior. • Ir a la pantalla de inicio.

Diseño

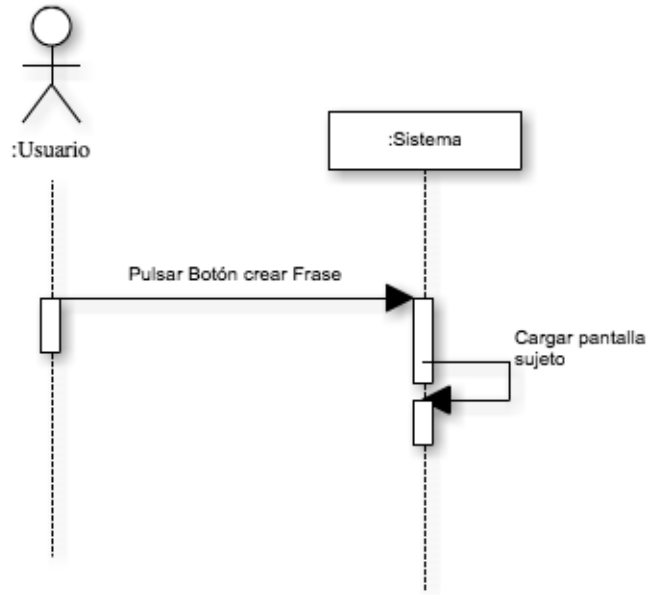
3. DISEÑO

3.1. DISEÑO DE LOS CASOS DE USO

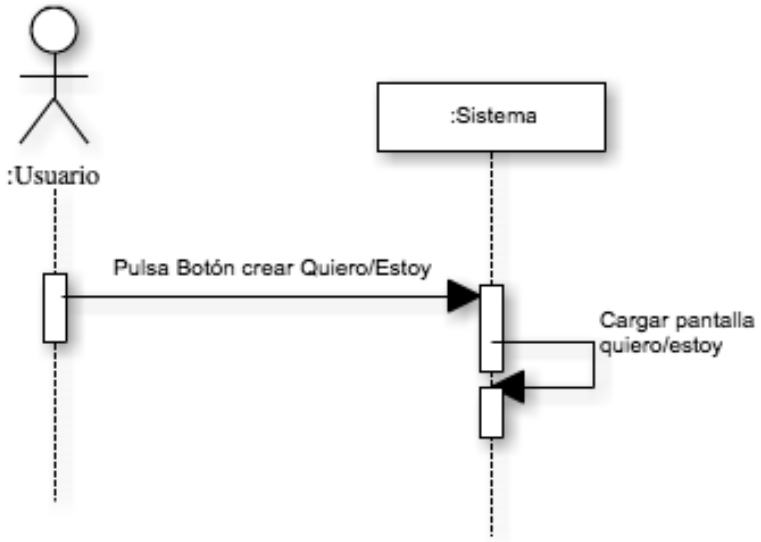
Se analizará cada caso de uso, preparando el diseño para su implementación a través de Diagramas de Secuencia.

Diseño de casos de uso de la Pantalla Principal

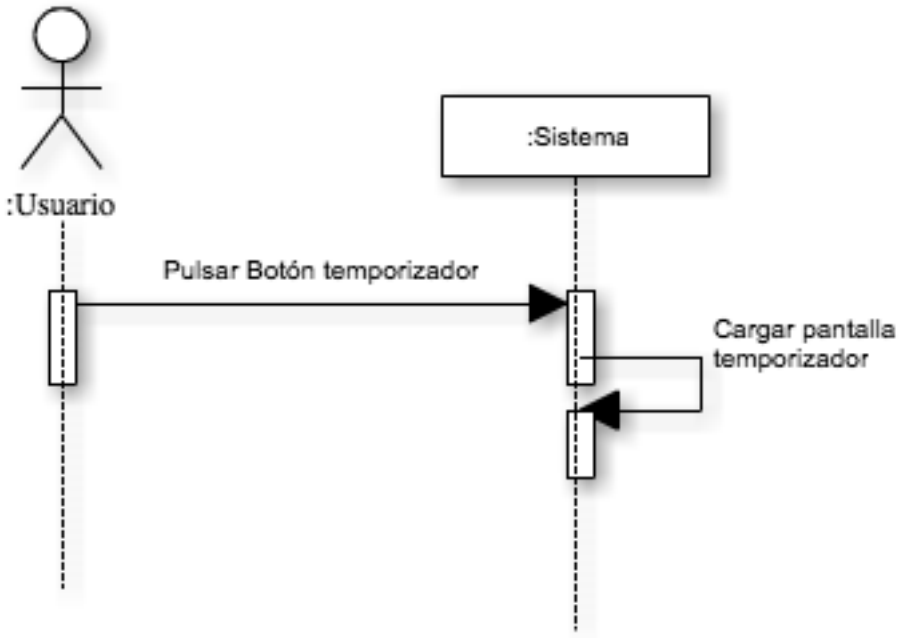
- IR A LA SECCIÓN DE CREAR FRASES:



- IR A LA SECCIÓN CREAR FRASES QUIERO/ESTOY:

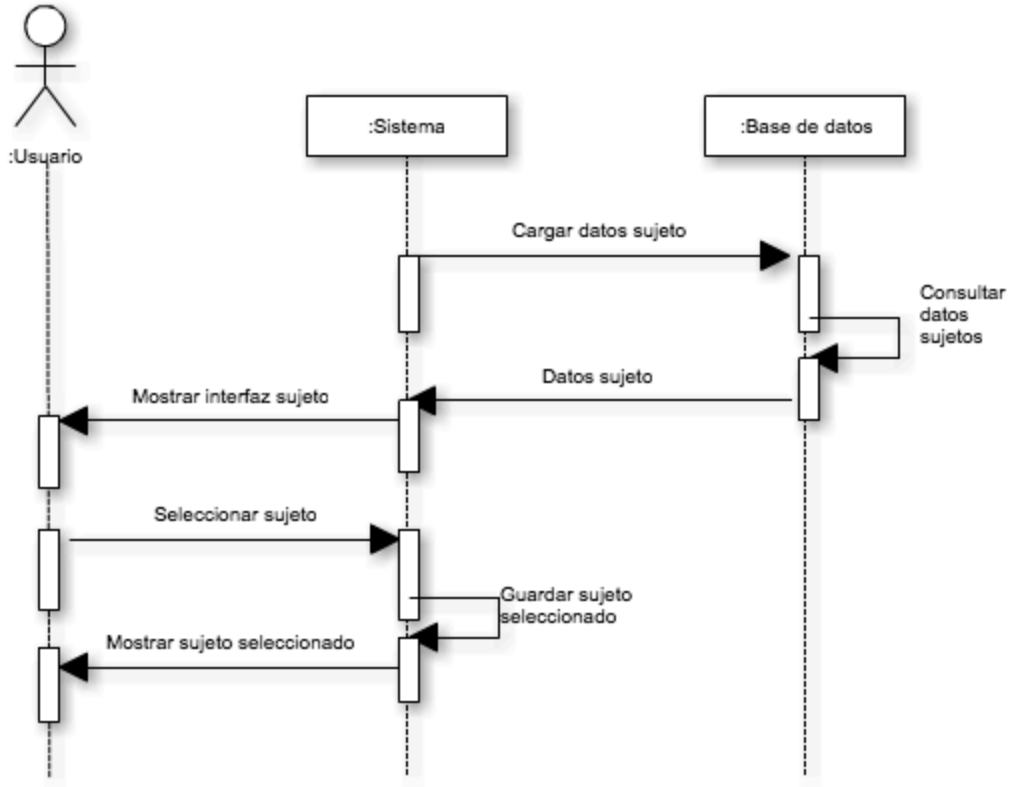


- IR A LA SECCIÓN DE PLANIFICACIÓN (TIEMPO):

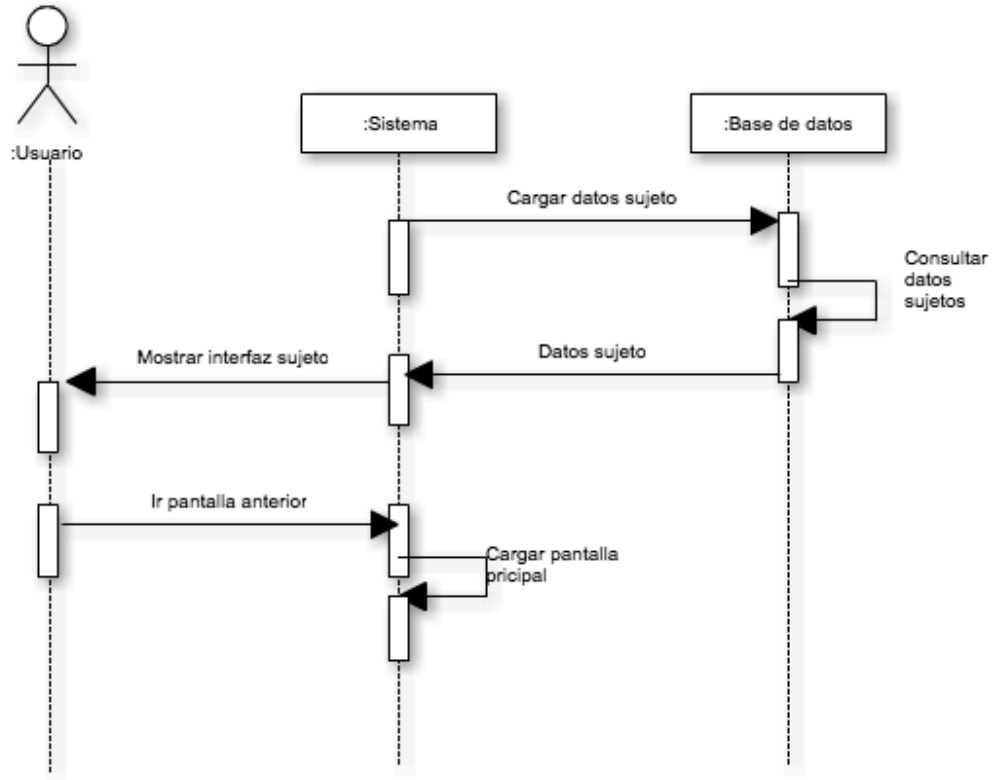


Diseño de casos de uso de la Pantalla Sujeto

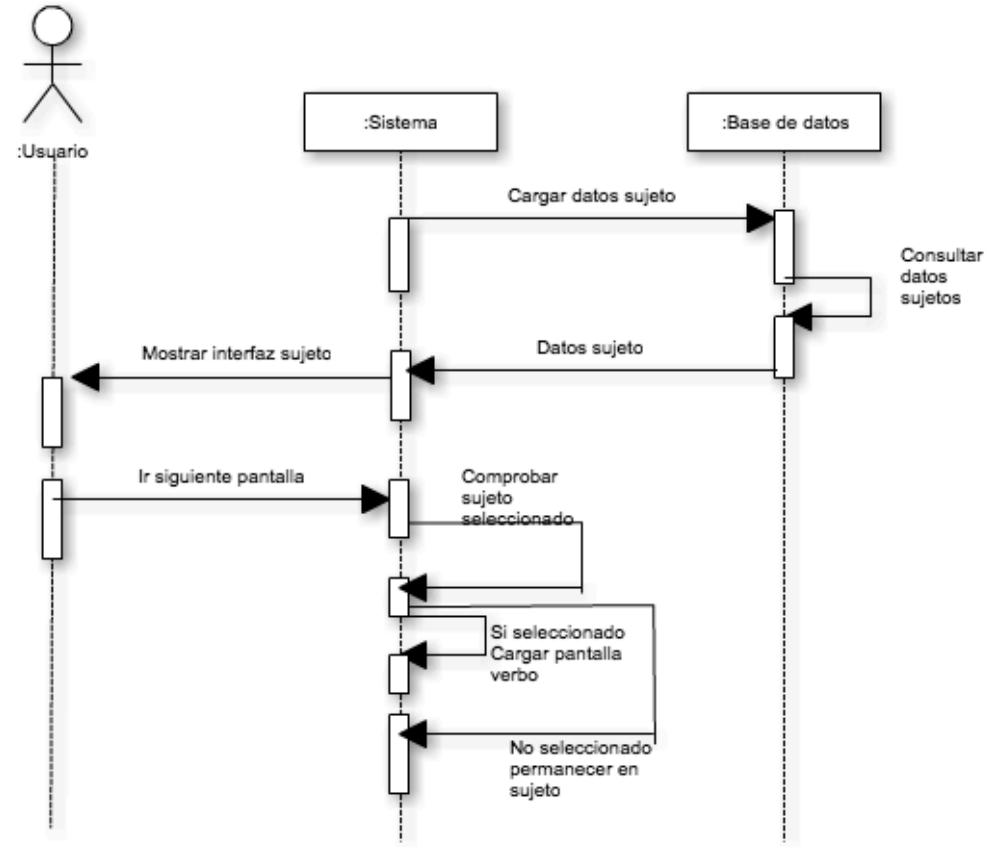
- SELECCIONAR UN SUJETO:



- IR A LA PANTALLA ANTERIOR:

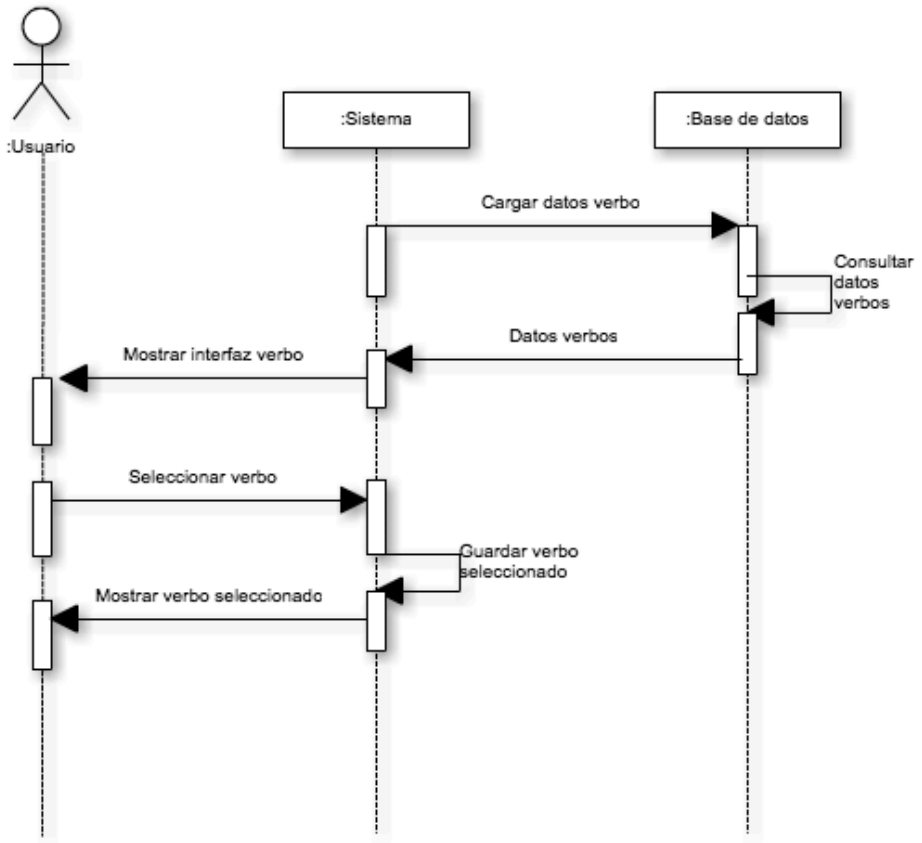


- IR A LA PANTALLA SIGUIENTE:

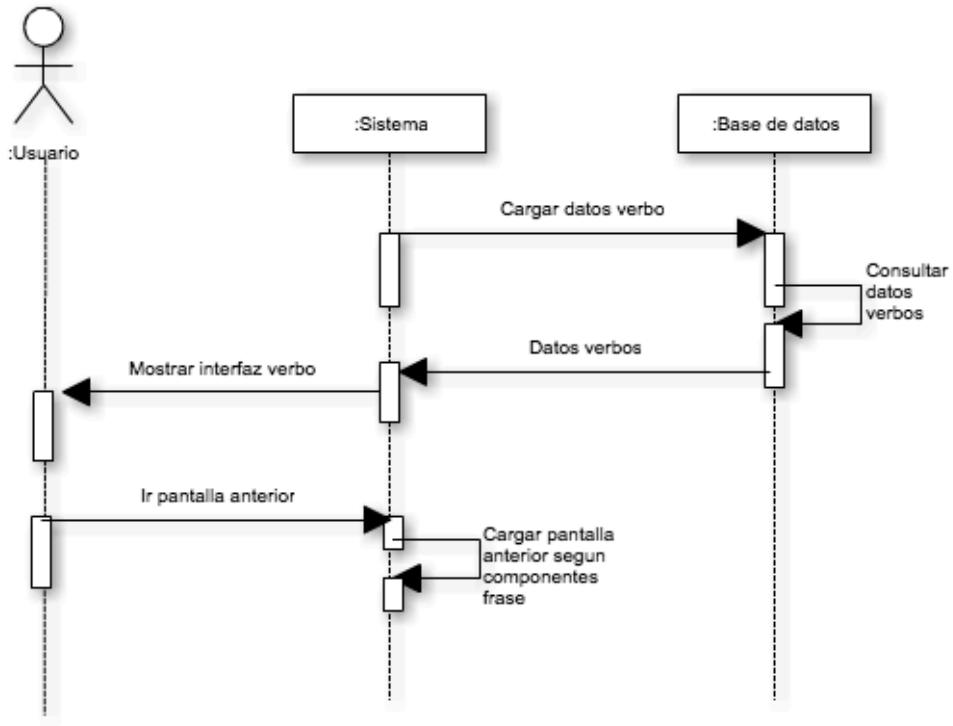


Diseño de casos de uso de la Pantalla Verbo

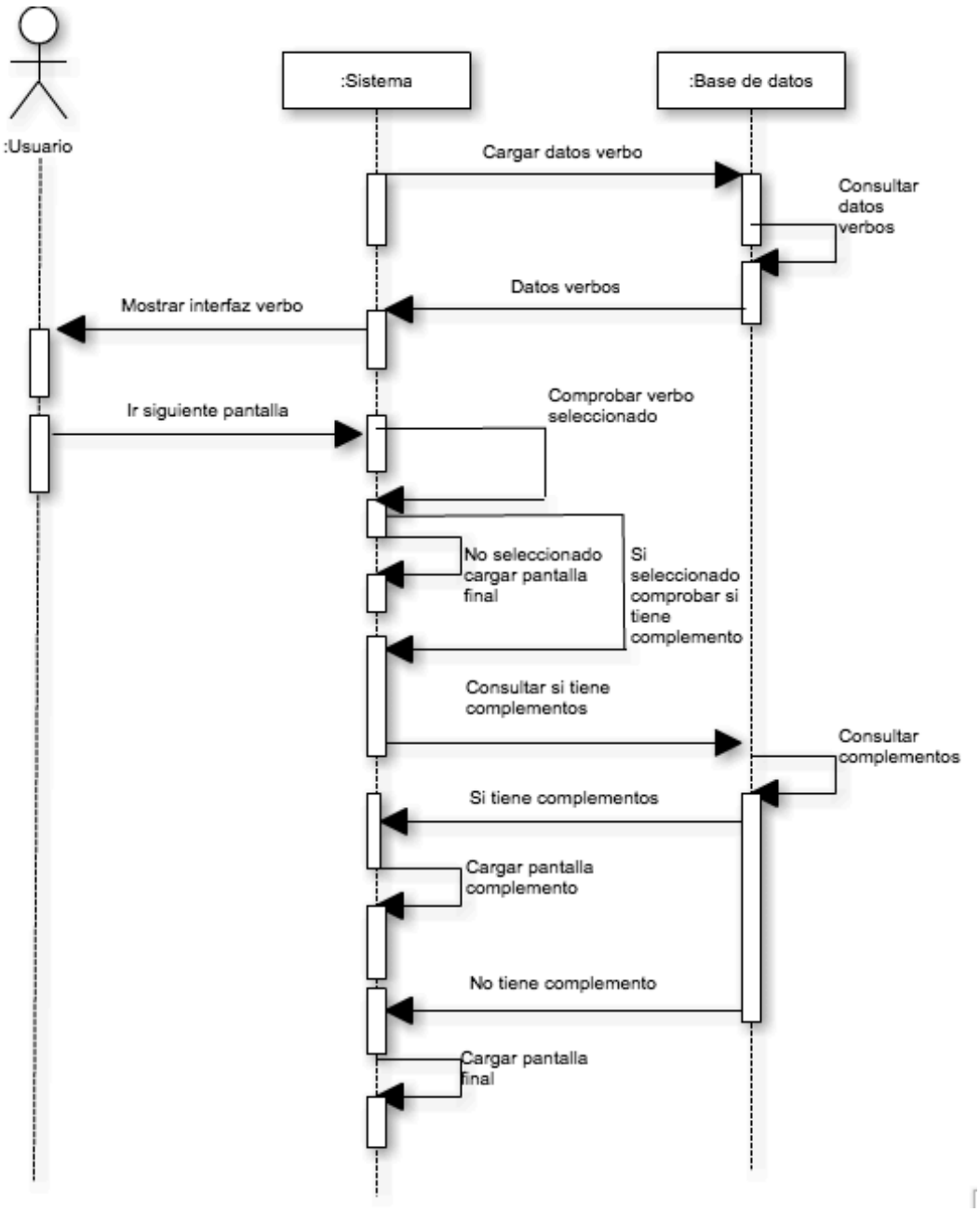
- SELECCIONAR UN VERBO:



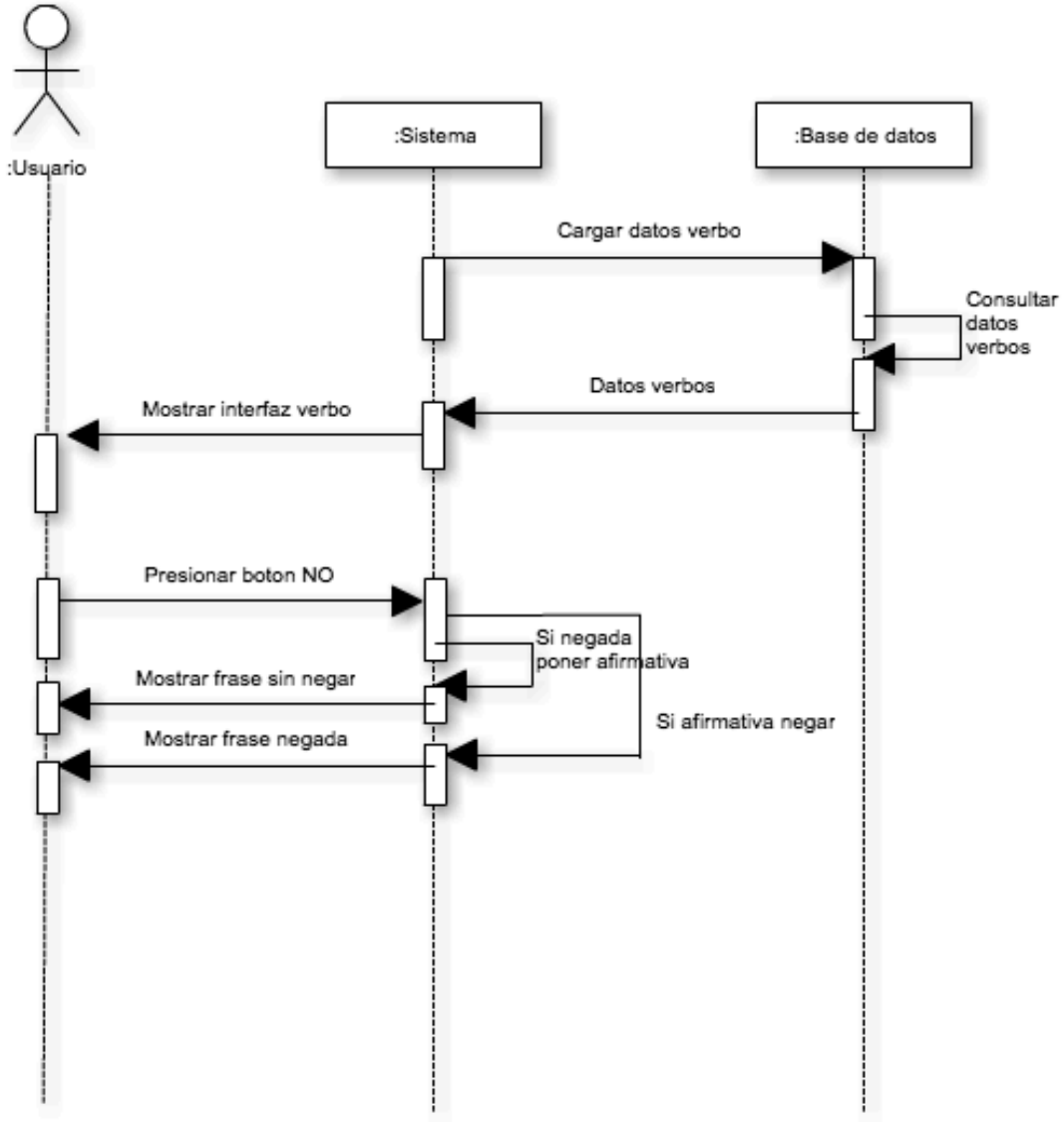
- IR A LA PANTALLA ANTERIOR:



- IR A LA PANTALLA SIGUIENTE:

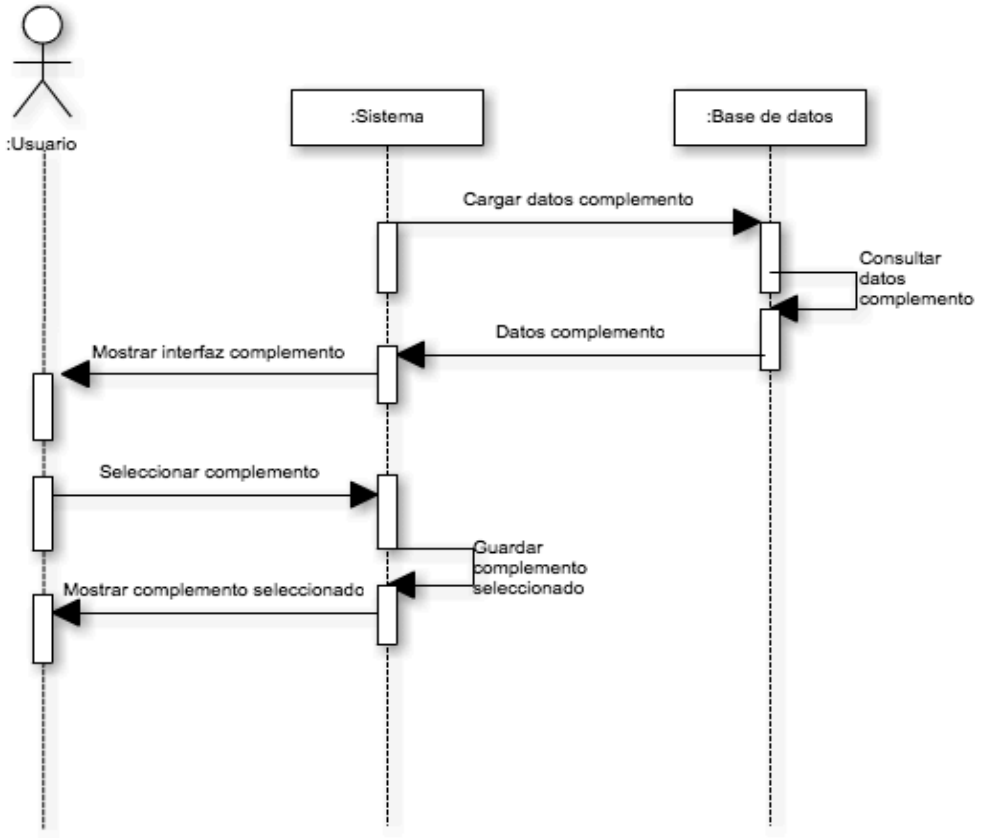


- NEGAR LA FRASE CREADA:

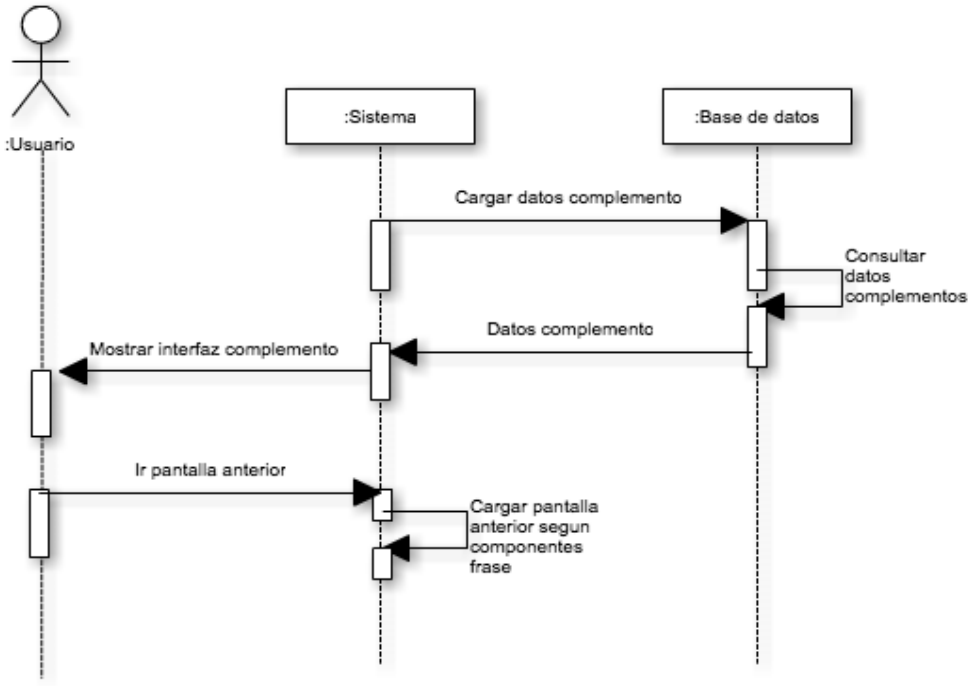


Diseño de casos de uso de la Pantalla Complemento

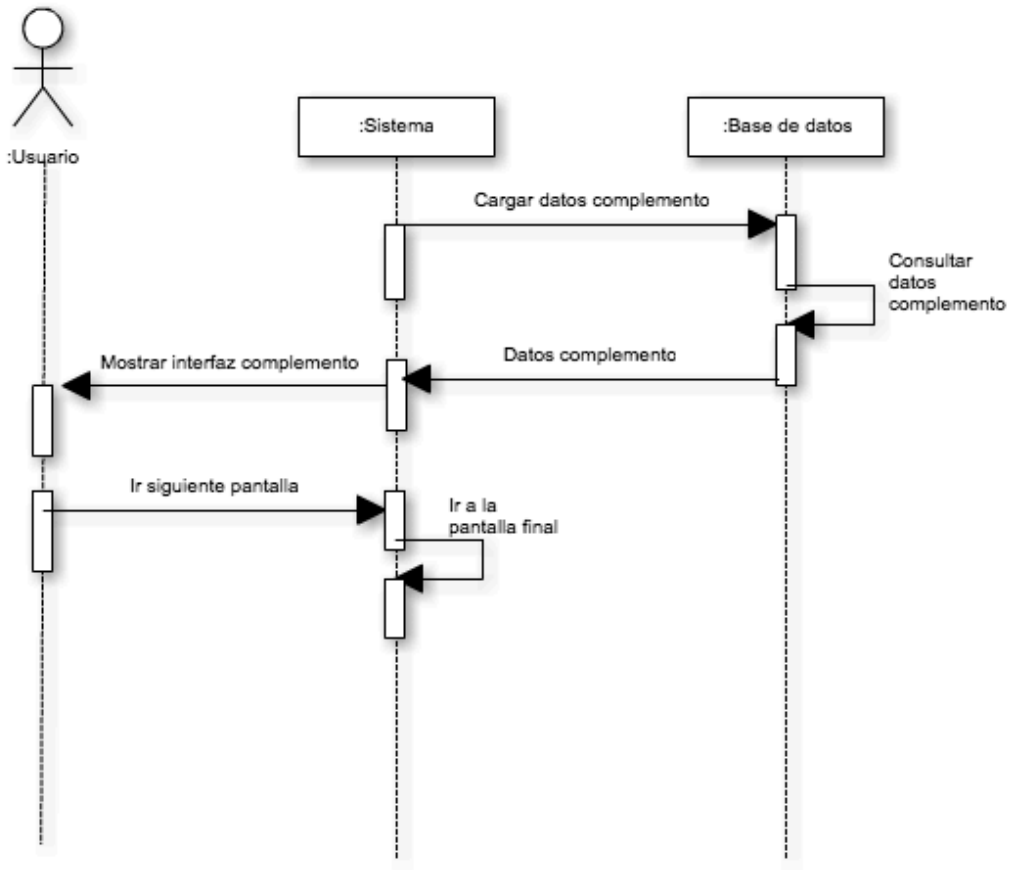
- SELECCIONAR UN COMPLEMENTO:



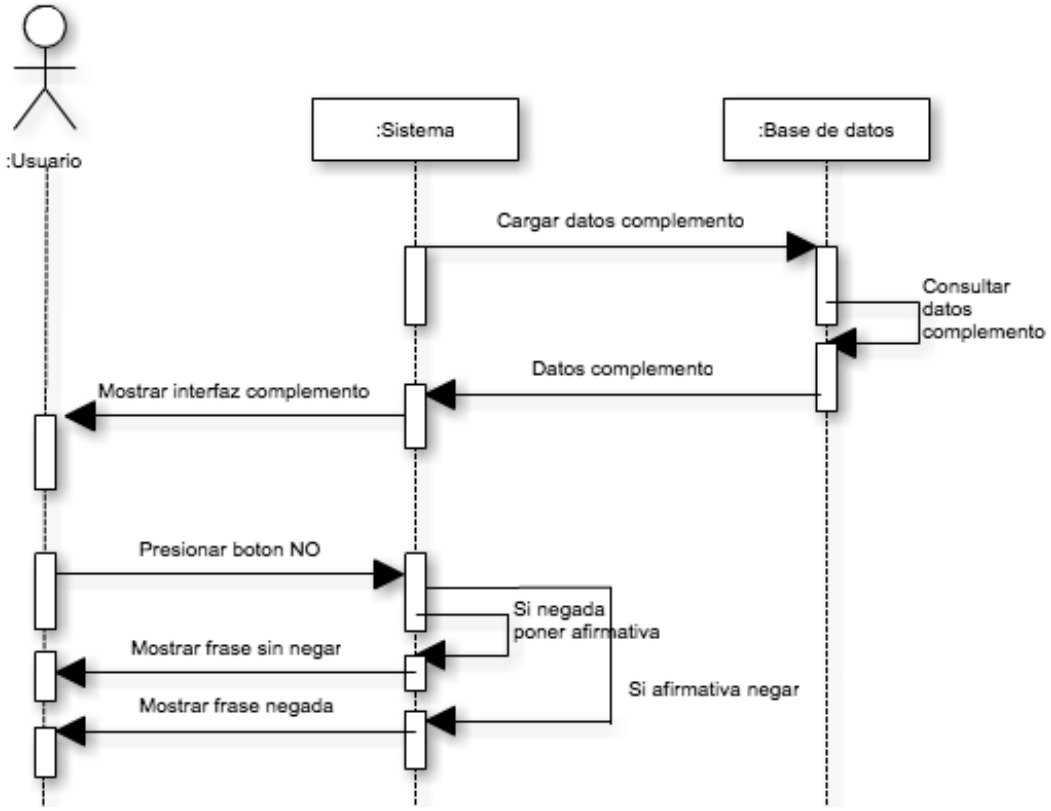
- IR A LA PANTALLA ANTERIOR:



- IR A LA PANTALLA SIGUIENTE:

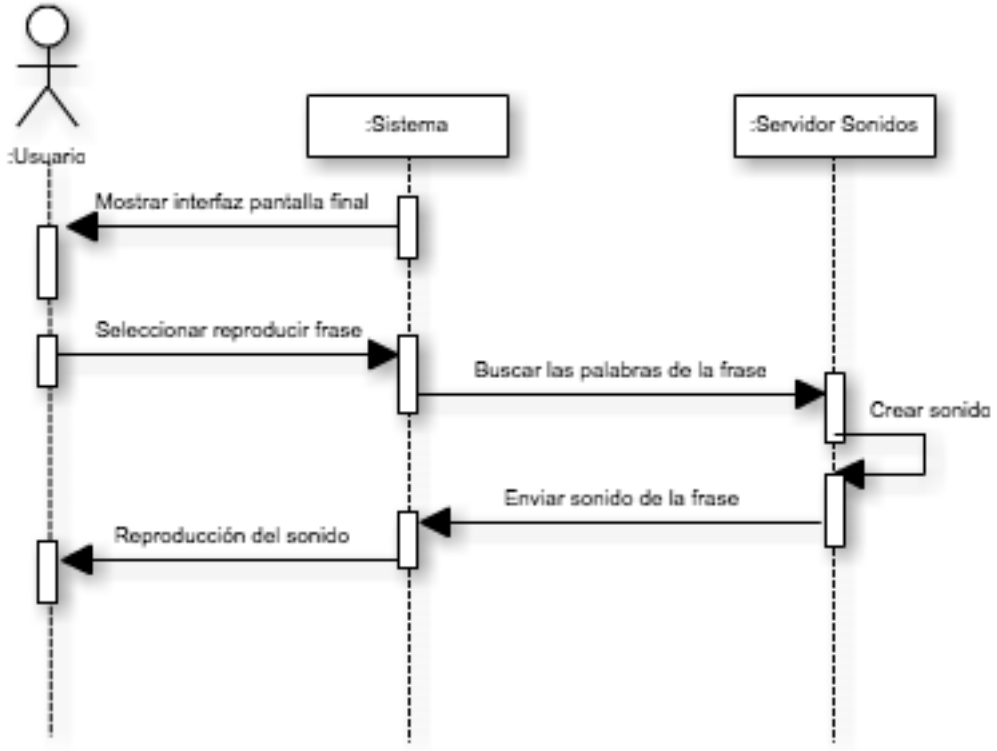


- NEGAR LA FRASE CREADA:

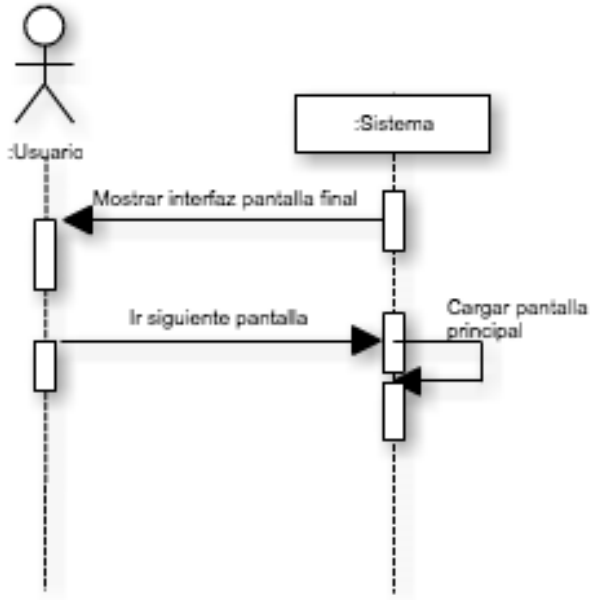


Diseño de casos de uso de la Pantalla Final

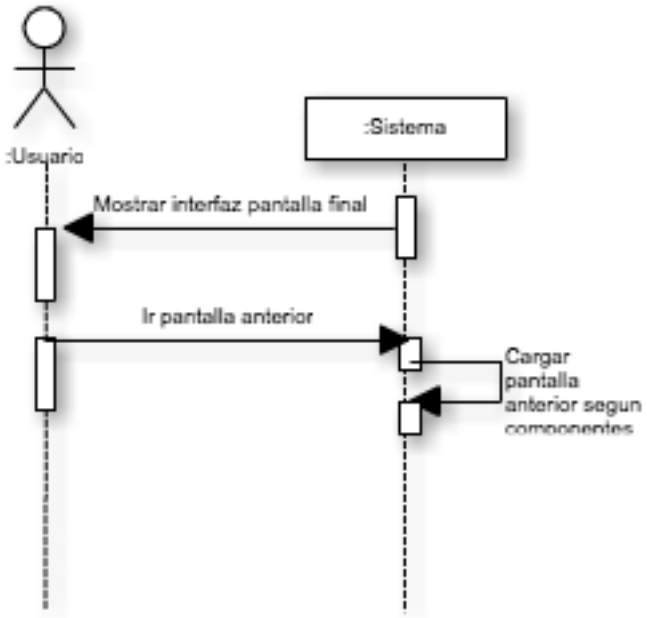
- REPRODUCIR LA FRASE:



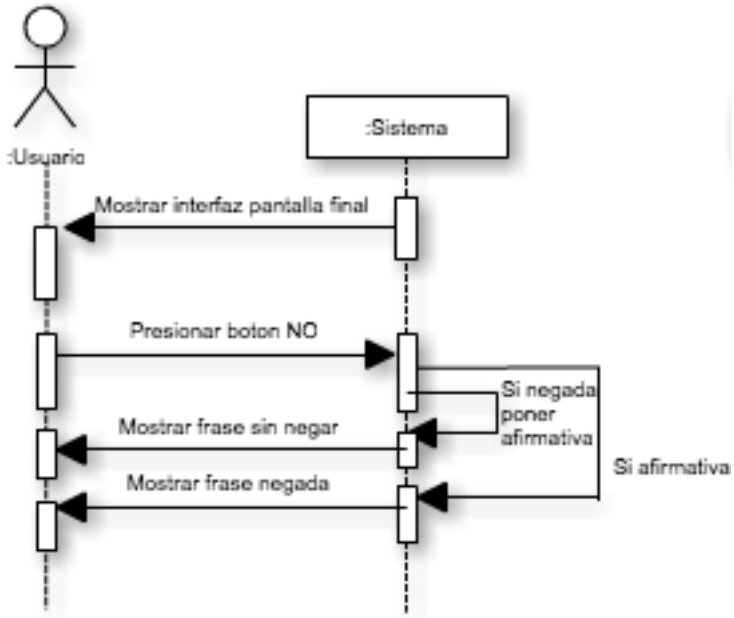
- IR A LA PANTALLA SIGUIENTE (INICIO):



- IR A PANTALLA ANTERIOR:

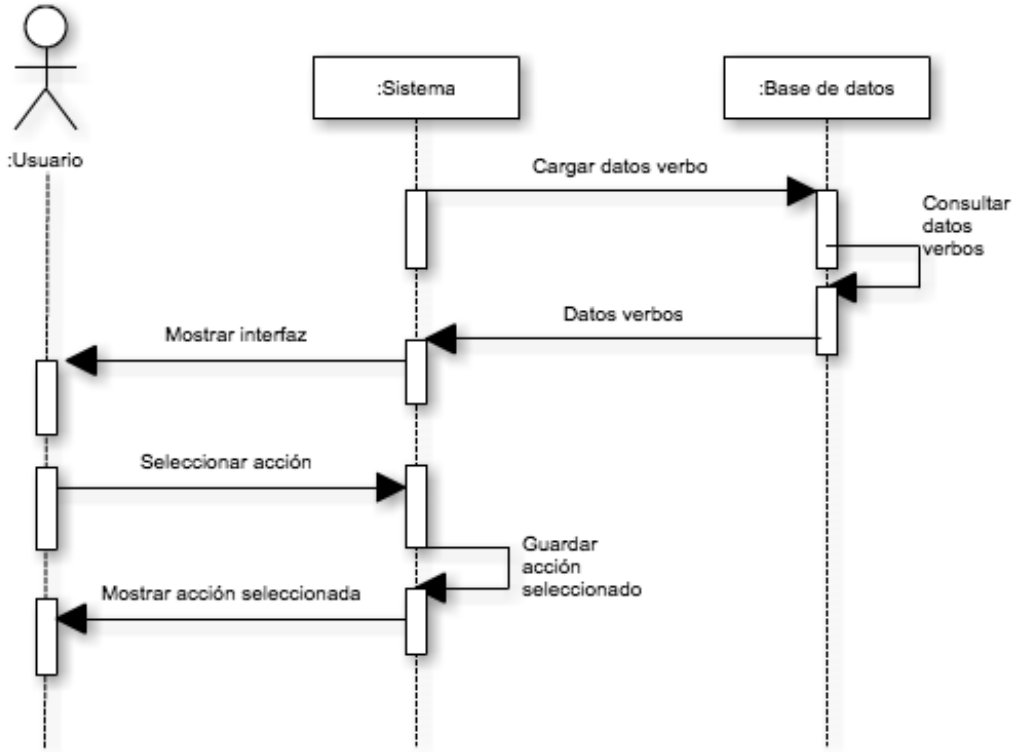


- NEGAR LA FRASE CREADA:

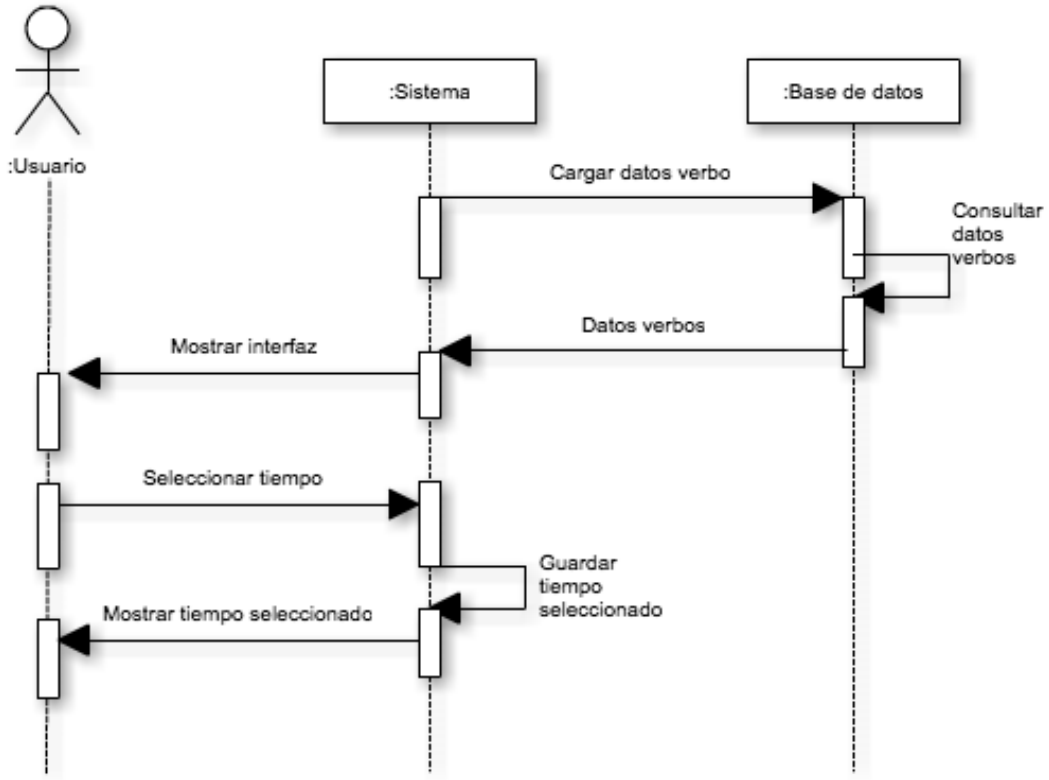


Diseño de casos de uso de la Pantalla Planificador

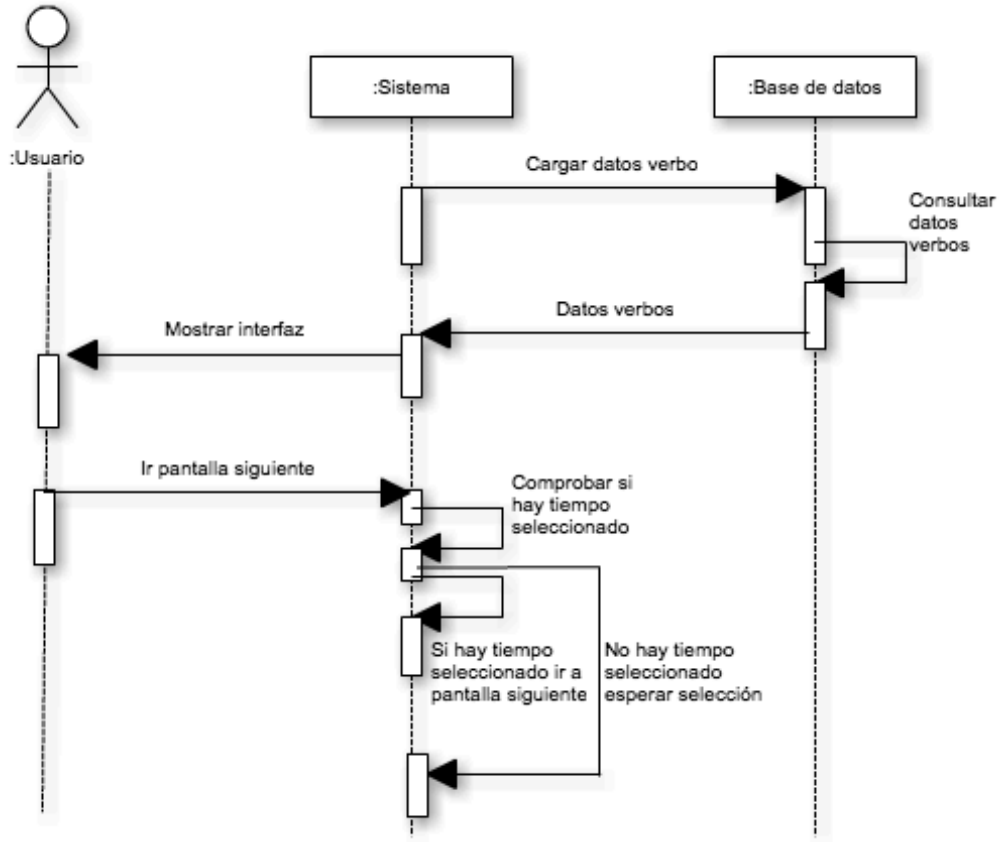
- SELECCIONAR ACCIÓN:



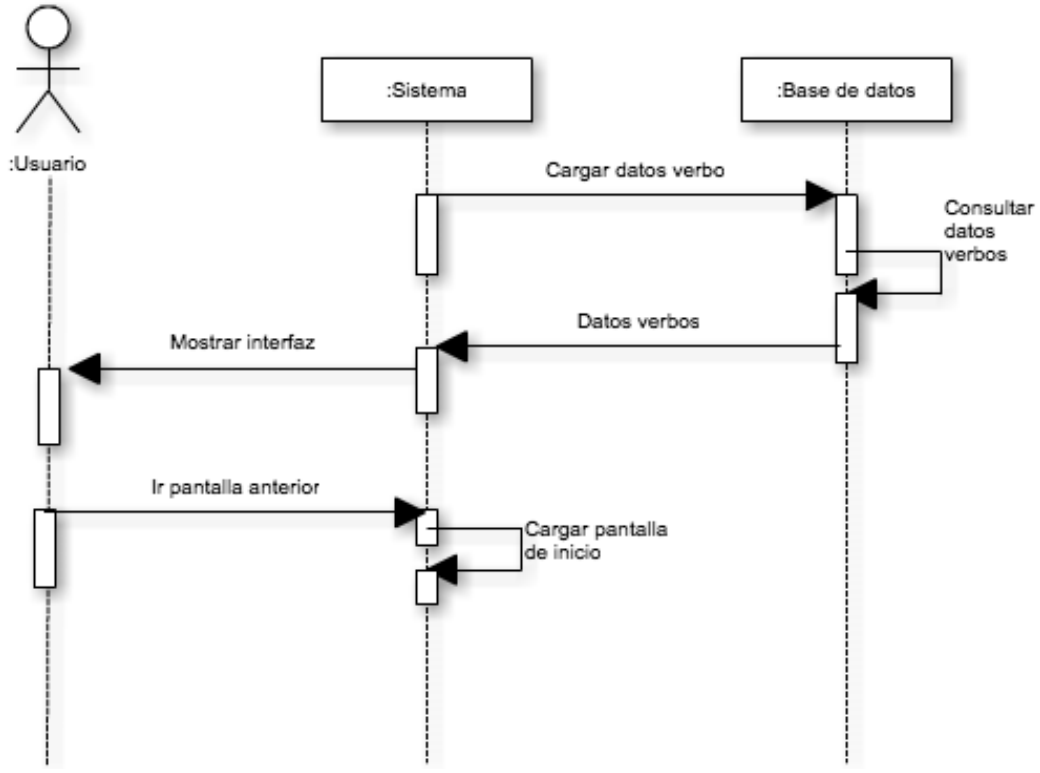
- SELECCIONAR TIEMPO:



- IR A LA PANTALLA SIGUIENTE:

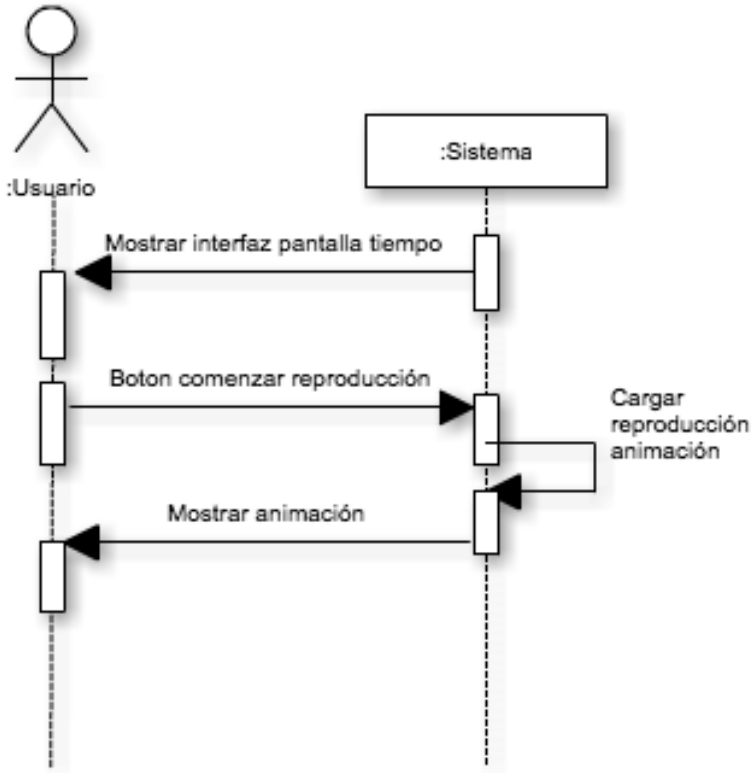


- IR A LA PANTALLA ANTERIOR:

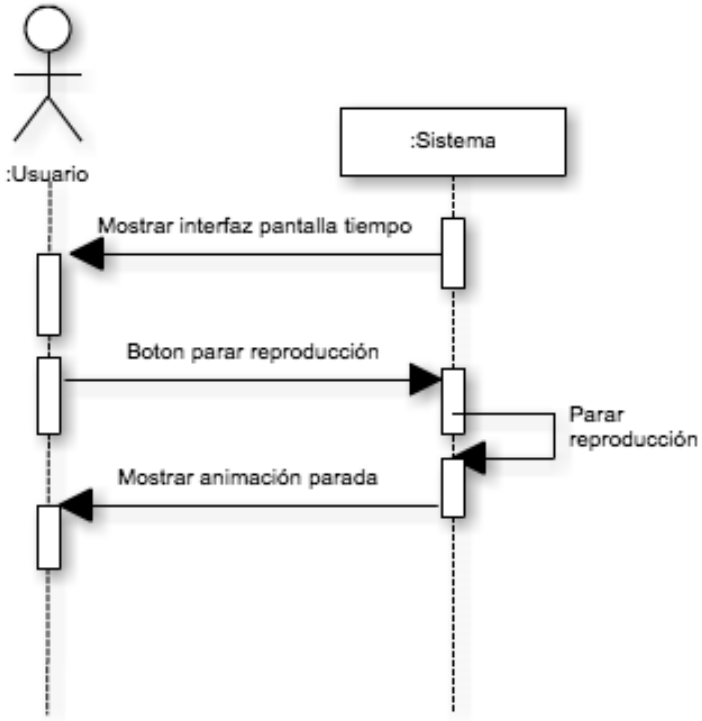


Diseño de casos de uso de la Pantalla Tiempo

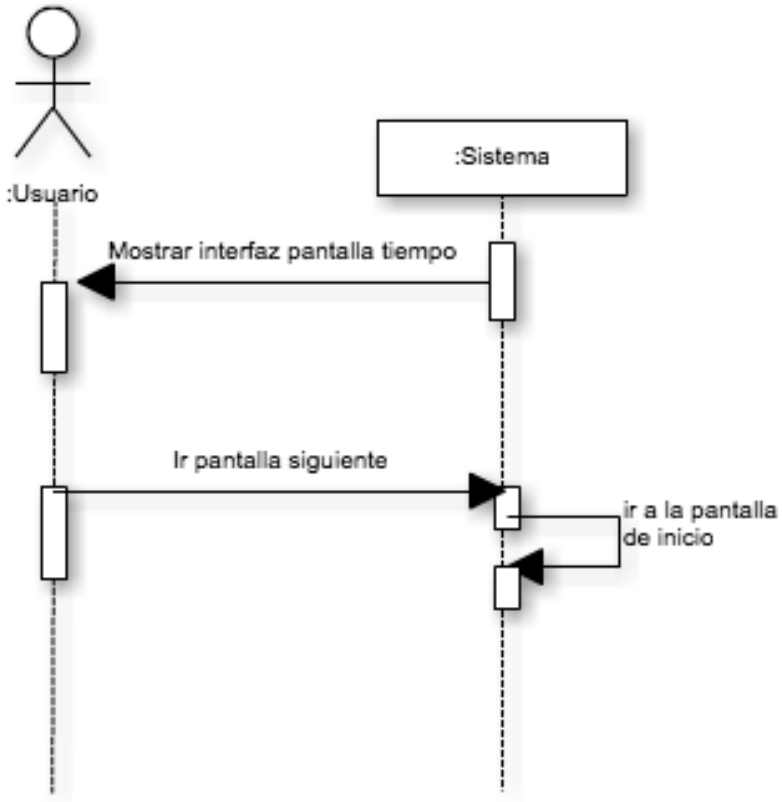
- REPRODUCIR ANIMACIÓN TIEMPO:



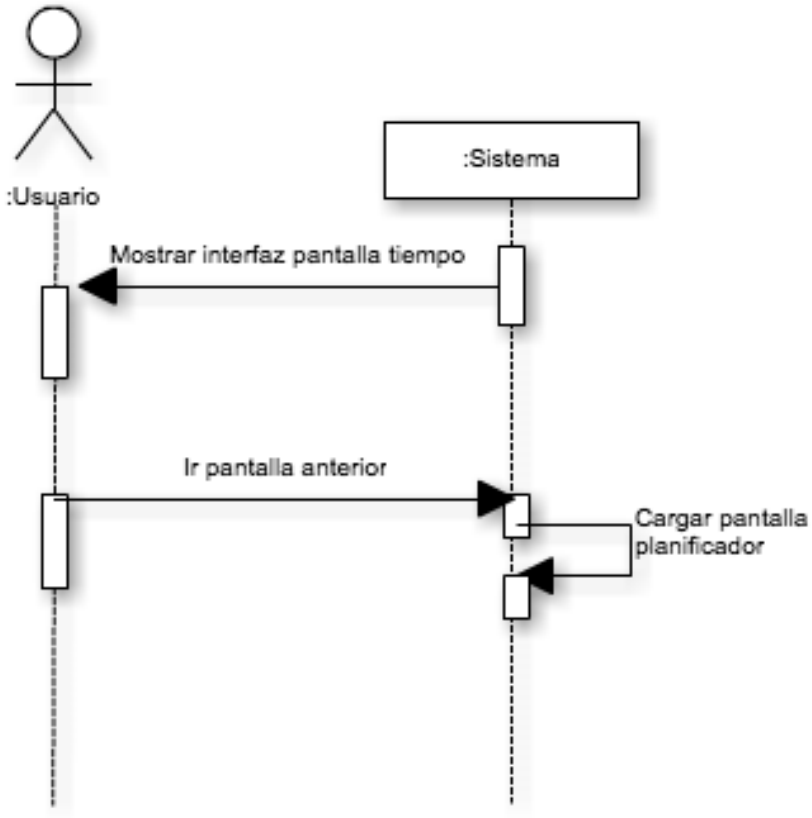
- PARAR REPRODUCCIÓN ANIMACIÓN TIEMPO:



- IR A LA PANTALLA SIGUIENTE (INICIO):



- IR A LA PANTALLA ANTERIOR:



3.2. CLASES DE LA APLICACIÓN

A continuación, se va a explicar las clases utilizadas para el diseño y posterior implementación de la aplicación. Las aplicaciones de Apple se basan en el patrón de diseño Modelo Vista Controlador (*MVC - Model Controler View*), de la programación orientada a objetos.

Como el modelo indica divide la aplicación en "3 campos":

- **Modelo**: representación de los datos con los que se va a operar en nuestra aplicación.
- **Controlador**: responde a eventos, usualmente acciones del usuario e invoca peticiones al modelo o la vista. Es la lógica de la interfaz de usuario.
- **Vista**: Representación gráfica de la interfaz (ventanas y otros elementos).

Basándonos en lo anterior podemos dividir las clases de la aplicación en esos 3 grupos expuesto.

Tendríamos por un lado las clases de Modelo, son aquellas que representan a los datos de la aplicación. Serán aquellas clases creadas como *NSObject* que nos servirán para acceder a los datos de la Base de Datos, a la que se accederá desde el *AppDelegate* de la aplicación.

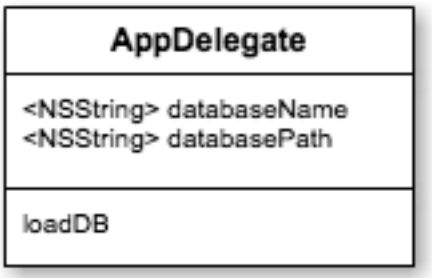
Por otro lado, estarán las clase de la parte del Controlador serán todas las vistas de la aplicación de la clase *UIViewControler*.

Y por último tendremos la Vista, en este caso serán los archivos .xib que correspondan a cada clase controlador, son los archivos donde tenemos todos los elementos que aparecen dentro del interfaz (*UIImageView*, *UICollectionView*, *UIPickerView*...).

CLASE APPDELEGATE

En primer lugar vamos a hablar de la clase *AppDelegate* de la aplicación que controlará la aplicación. En nuestro caso será además la encargada de realizar la conexión a la base de datos (BBDD.sqlite).

También controlará la variable de la clase Global, que utilizaremos para almacenar los valores de las variables necesaria para poder crear la frase, ya que la única forma de tener acceso a ellas es a través de la clase controlador (*AppDelegate*).

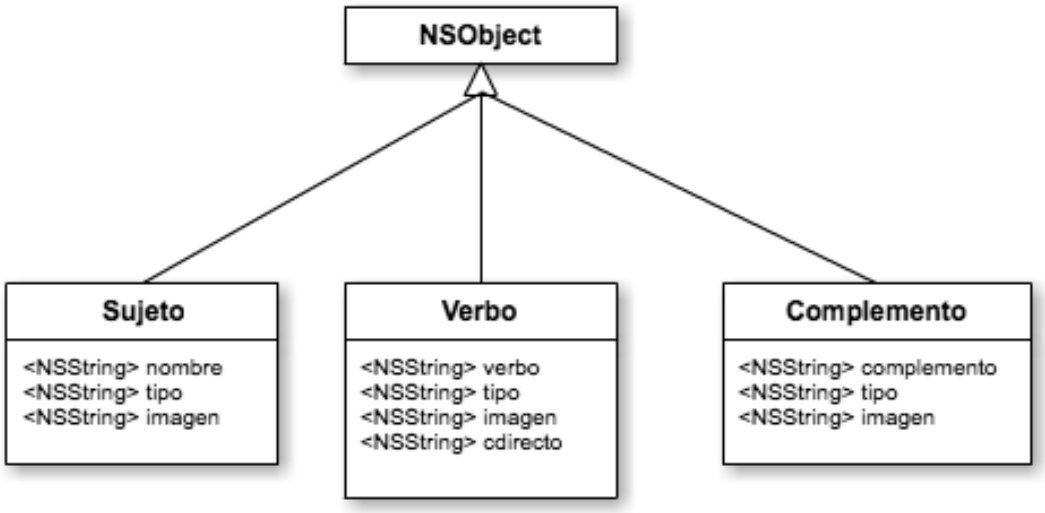


La clase Global donde tendremos la información necesaria para la formación de la frase.

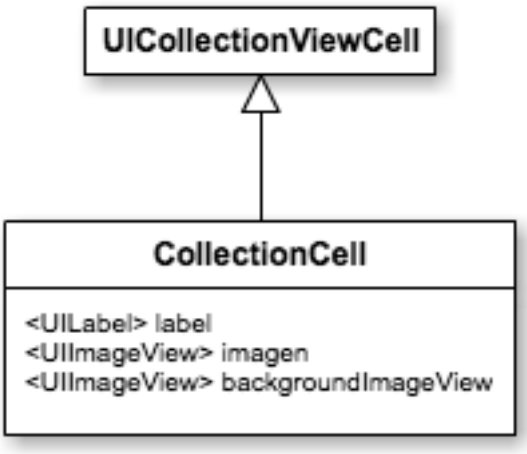
CLASES DE MODELO

Dentro de las clases de modelo encontramos:

- Por un lado, las clases necesarias para guardar los datos que vamos a obtener de la base de datos (serán de tipo *NSObject*).



- Por otro lado, está la clase CollectionCell que nos sirve para almacenar los datos que mostraremos en las celdas del *UICollectionView*, donde se mostraran las imágenes con la palabra que se representa (verbo, sujeto, complemento). Para estas celdas hay una clase ya predefinida que es *UICollectionViewCell*.



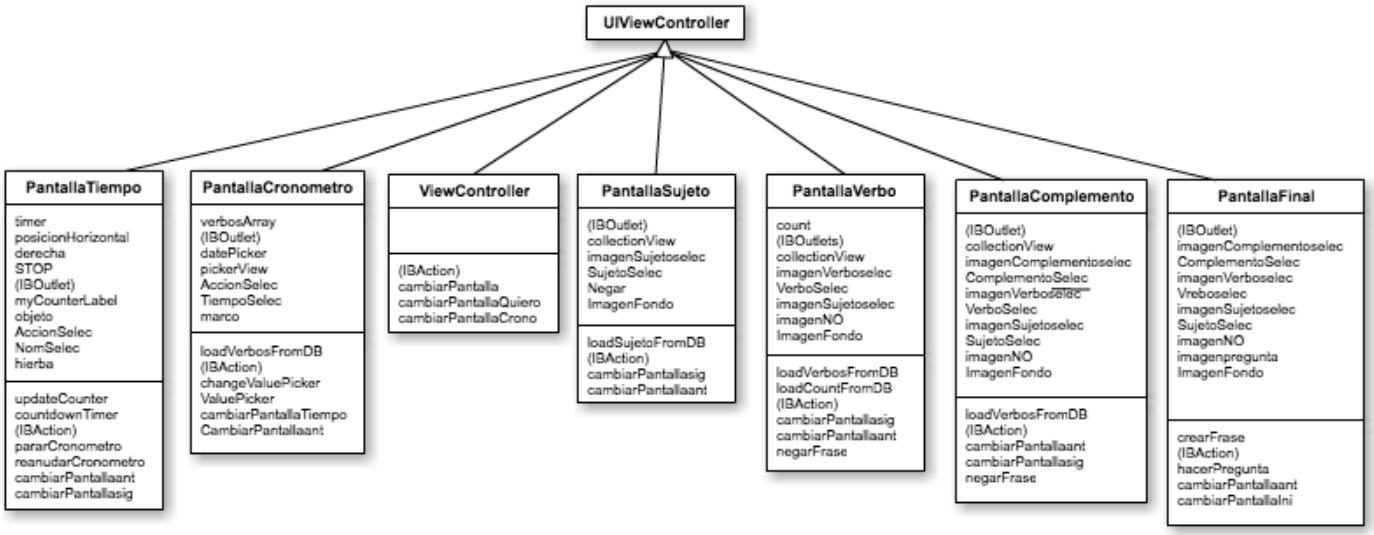
Como se puede observar las celdas estarán compuestas por 2 imágenes (*UIImageView*), una será de fondo y la otra para la imagen que represente la palabra y por último, una etiqueta (*UILabel*).

Con esto y la base de datos tendríamos toda la parte de datos de la aplicación, y podemos pasar a ver la parte del controlador de los datos.

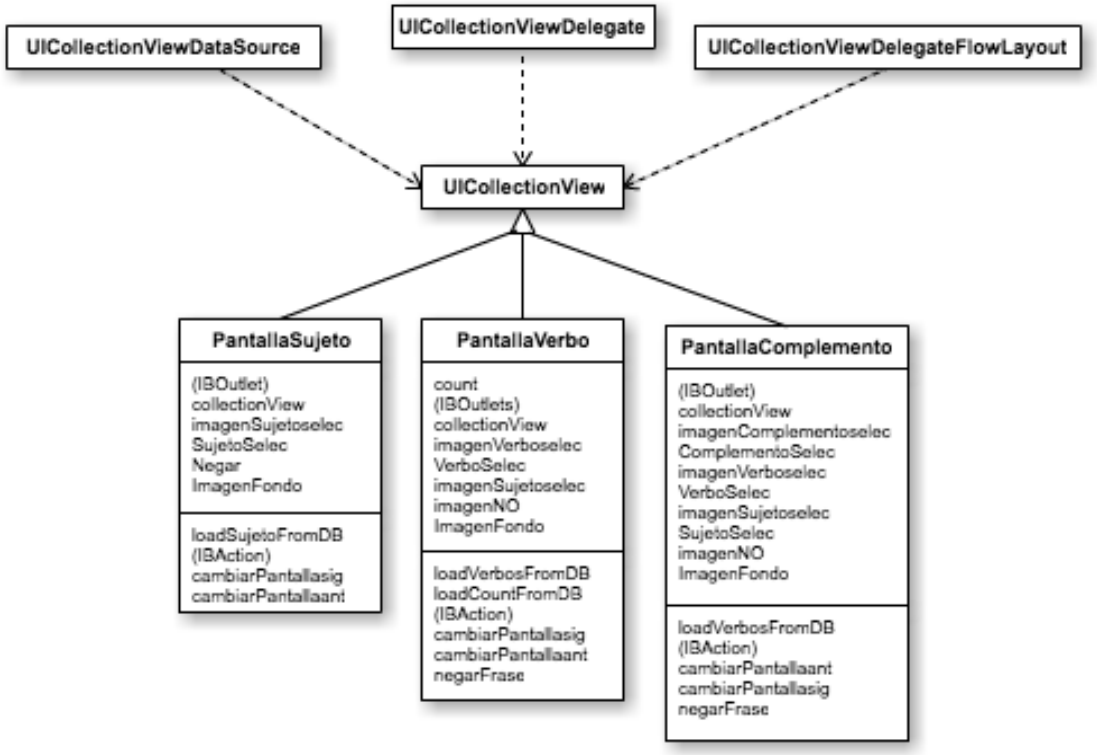
CLASES CONTROLADOR

Dentro de este apartado tendremos las clases controlador que se corresponderán con las vistas de la aplicación, en nuestro caso tendremos una clase por cada interfaz de la aplicación como ya hemos dicho con su correspondiente vista que más tarde explicaremos con más detalle.

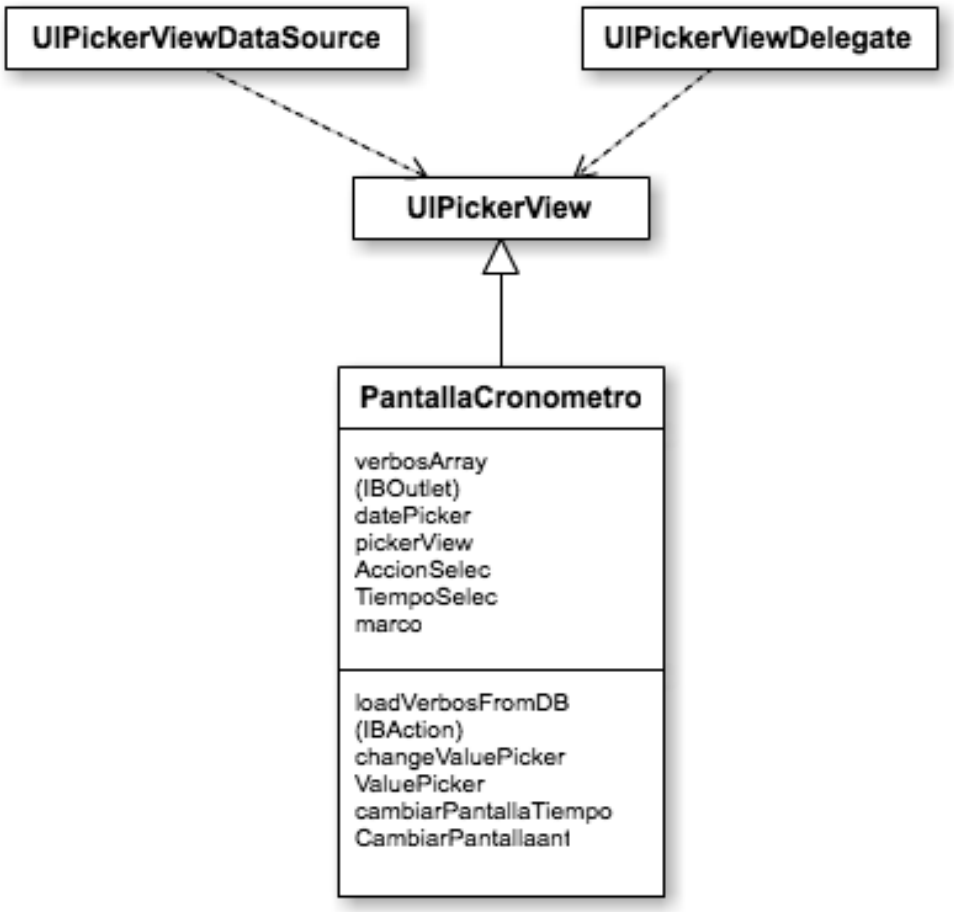
Nos encontramos con las siguientes subclases *UIViewController*.



Además las clases PantallaSujeto, PantallaVerbo y PantallaComplemento, necesitarán utilizar métodos de la clase *UICollectionView* ya que utilizan objetos de esa clase como parte de sus vistas.

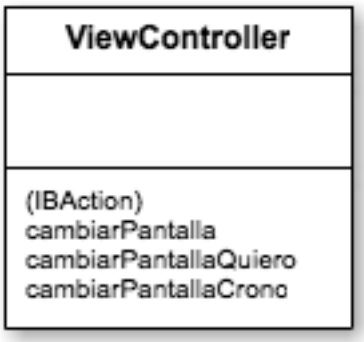


Por otro lado la clase PantallaCronometro, utilizar un objeto de la clase *UIPickerView*.



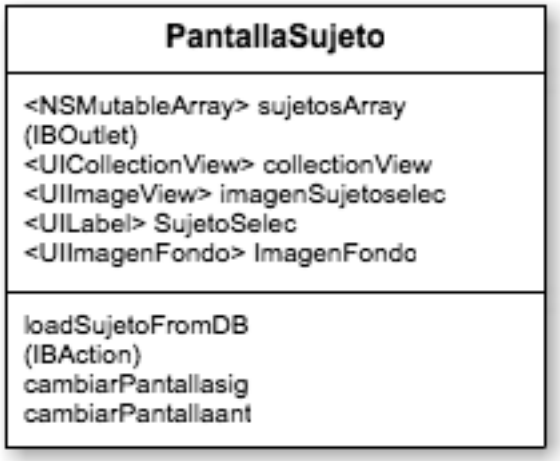
A continuación aparecerán las clases utilizadas con sus atributos y métodos.

Clase *ViewControler*, es la clase donde podremos elegir entre las distintas opciones de la aplicación, menú principal de la aplicación.

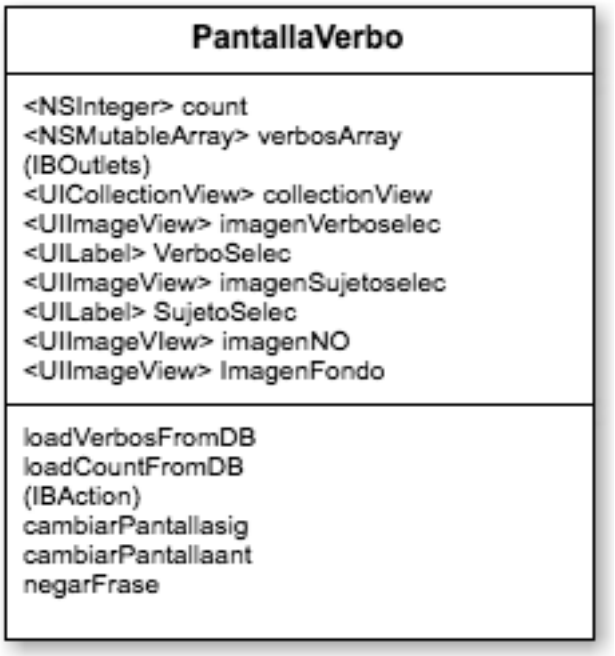


Si se ha elegido la opción de crear frase, iremos a PantallaSujeto, donde se haría la elección del sujeto de la frase. Si nos encontramos ante el tipo de frases

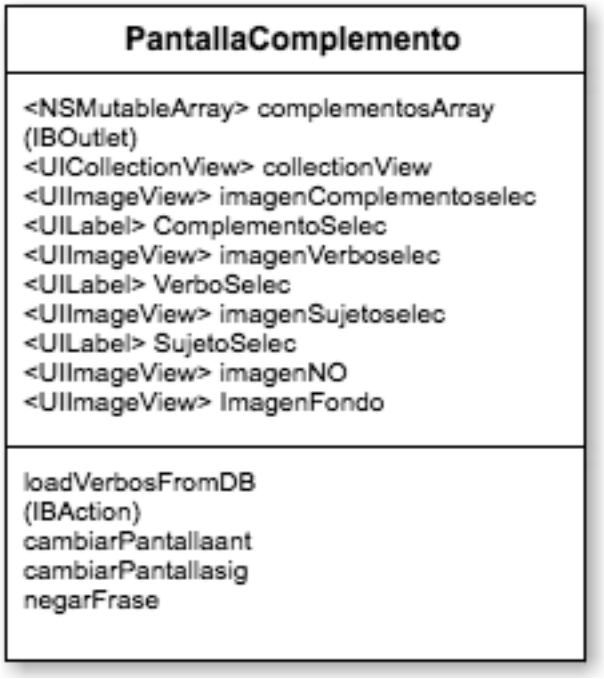
Quiero/Estoy, en esta pantalla se elegirá por entre esas dos palabras para el comienzo de la frase.



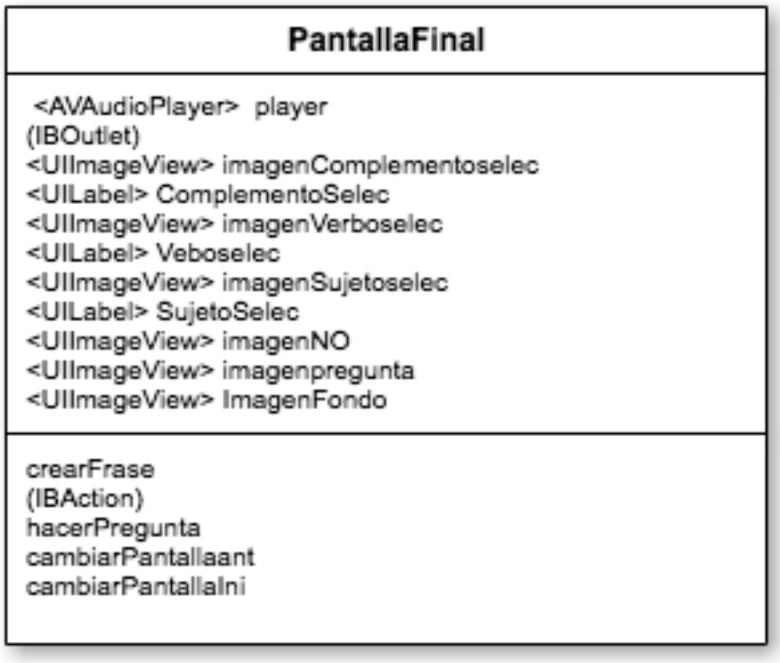
De PantallaSujeto por en botón cambiarPantallasig (metodo correspondiente al botón), nos llevará a la siguiente clase PantallaVerbo.



Después iríamos a la clase para la elección del complemento, PantallaComplemento.

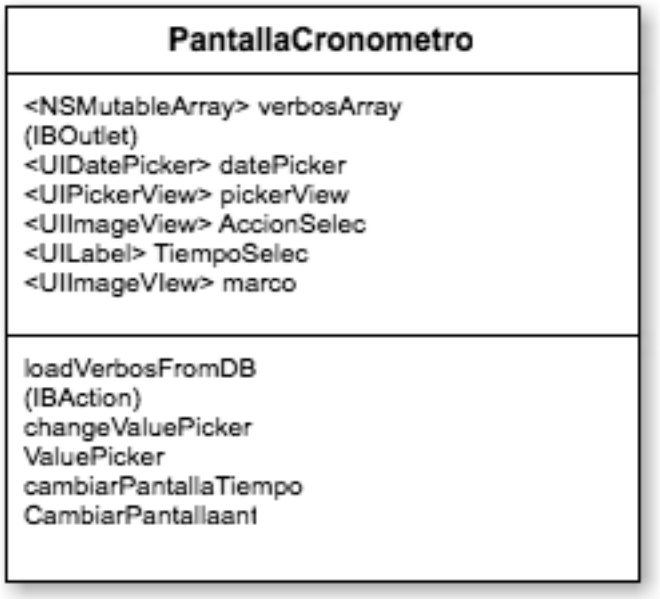


Por último se llegará a la PantallaFinal donde ya tendremos la frase creada, y se podrá reproducir.

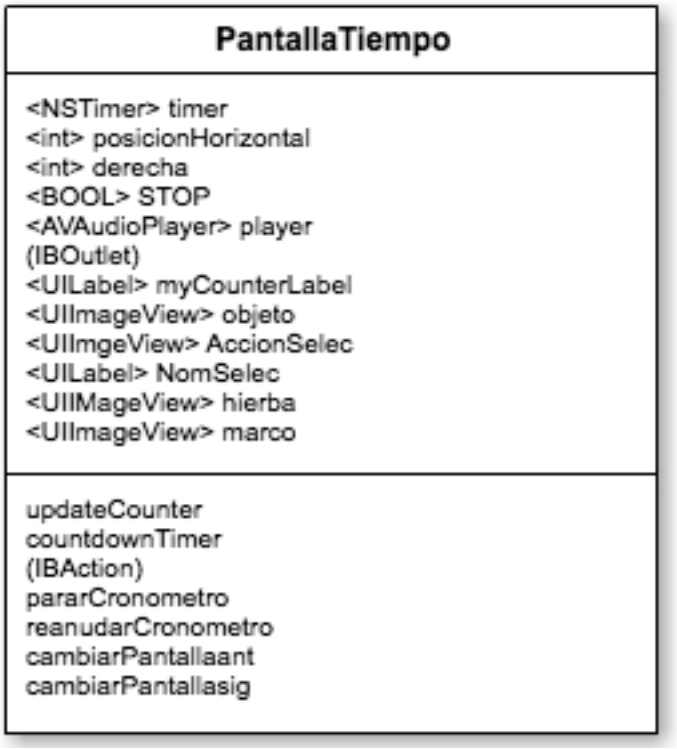


Para la reproducción de la frase tendremos que utilizar objetos de la clase *AVAudioPlayer*, que nos permitirá la reproducción del audio de las palabras que teníamos como texto de la frase creada (*Text to speach*).

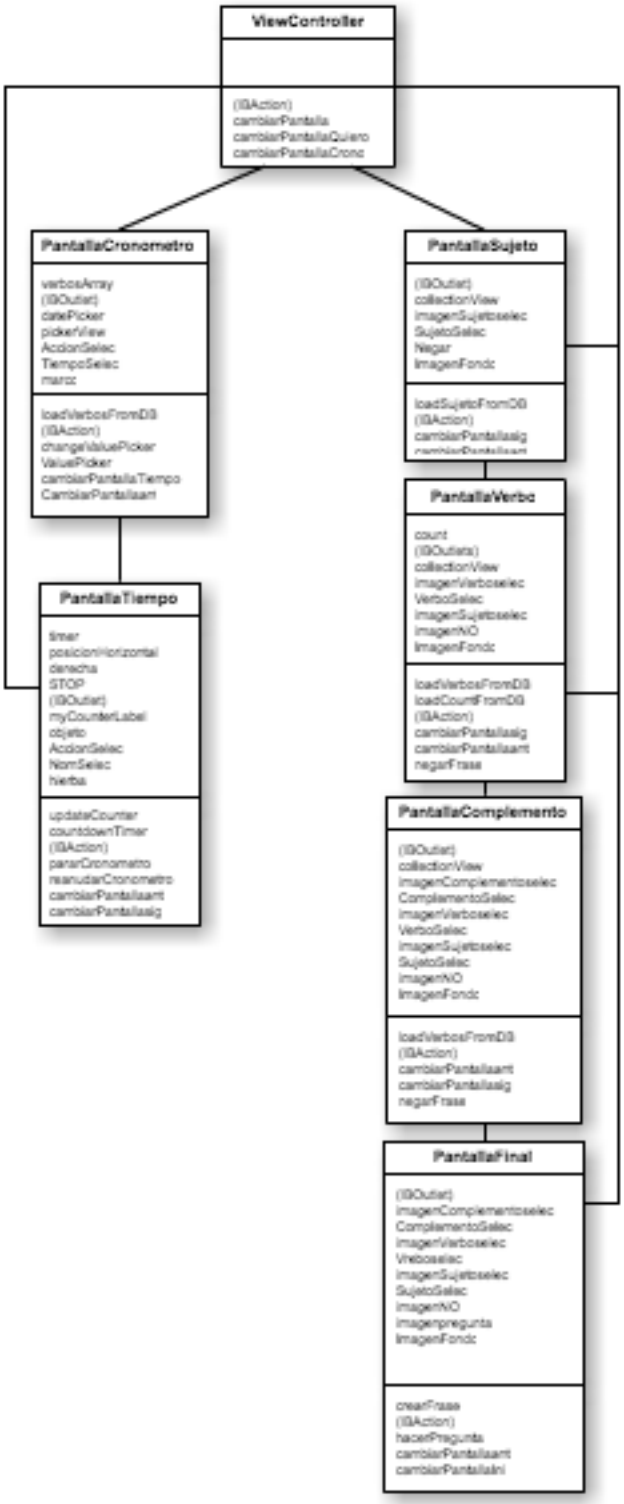
Por otro lado, si lo que se elige es la opción del cronómetro de las actividades. Iremos a la PantallaCronometro donde realizaremos la elección del tiempo y la actividad a realizar en ese tiempo.



Después la animación del cronometro del tiempo y el movimiento del objeto con el paso del tiempo se realizará en la PantallaTiempo.



Aquí se pueden observar todas las clases de control que tenemos en la aplicación.



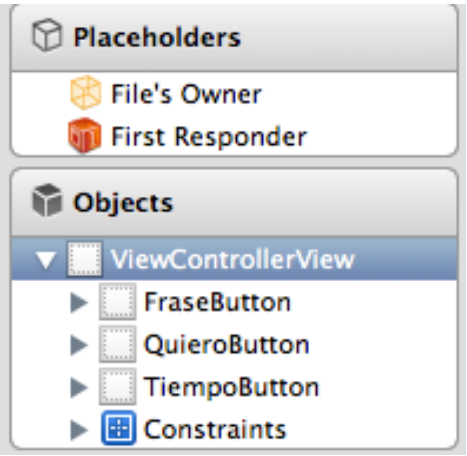
VISTAS DE LA APLICACIÓN

Como ya dije la parte de vistas se corresponde con los ficheros .xib creados para cada una de la clases controlador. En ellos aparecen todos los objetos que forman parte de la interfaz de usuario, todo aquello que vamos a poder ver en la pantalla de la aplicación (botones, imágenes, etiquetas, collectionView, pickers...).

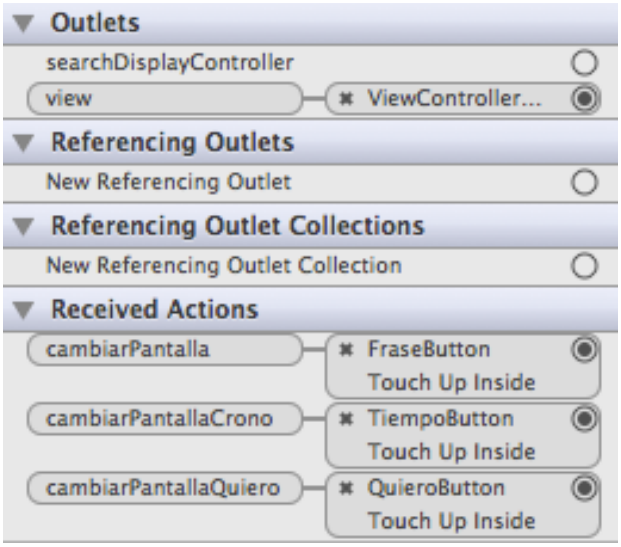
Vamos a mostrar para cada una de la clases controlador la composición de su vista correspondiente (.xib).

Clase ViewController

Estará compuesta por 3 botones como se puede observar a continuación (frase, quiero/estoy y tiempo).

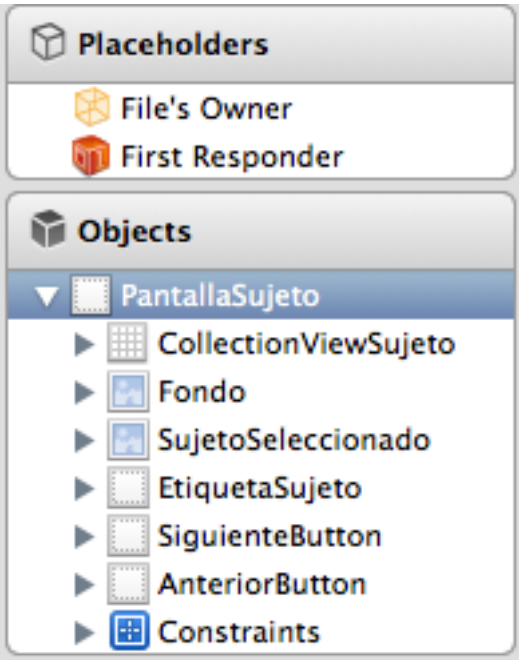


A cada uno de los objetos que aparecen se les asignará un comportamiento, que estará implementado como ya se ha dicho en la clase controlador. *File's Owner* nos permite asignar las acciones a realizar (*IBAction*).

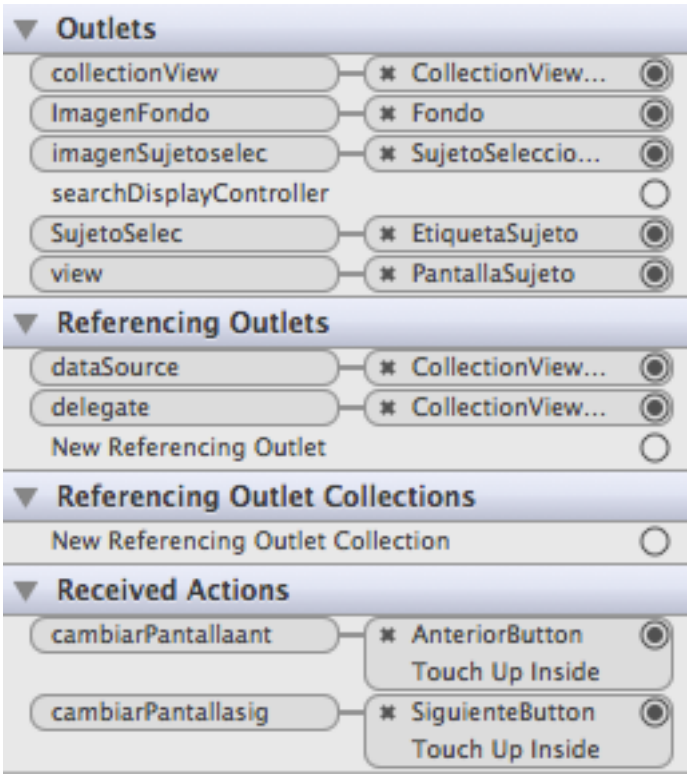


Clase PantallaSujeto

En la pantalla sujeto, donde elegiremos el sujeto si es frase normal o entre Quiero y estoy en el otro caso. Será un pantalla compuesta por imágenes, *UICollectionView*, etiquetas y botones.

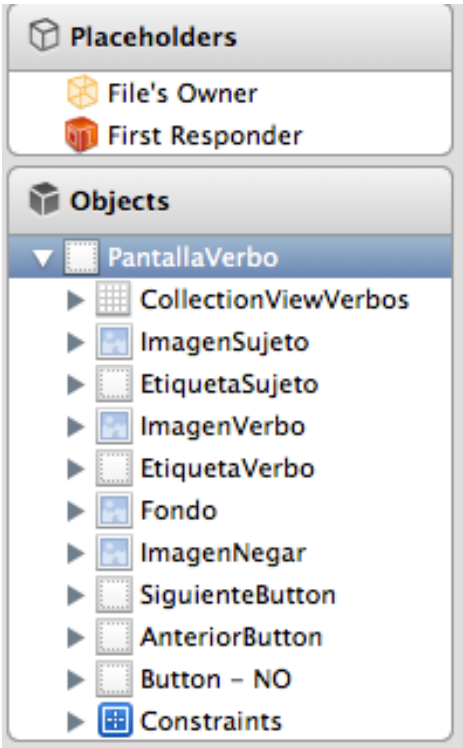


Las acciones asignadas por *File's Owner* quedarían del siguiente modo:

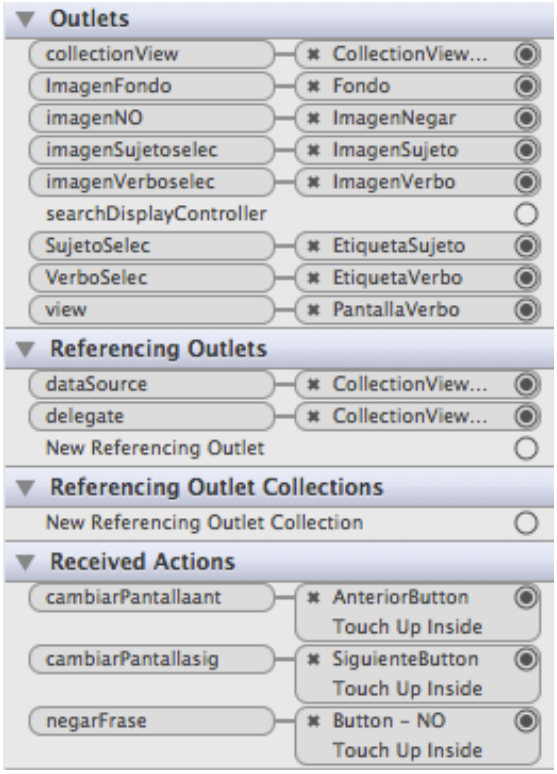


Clase PantallaVerbo

En el caso de la pantalla para la elección de verbo quedaría compuesta por los siguientes objetos:

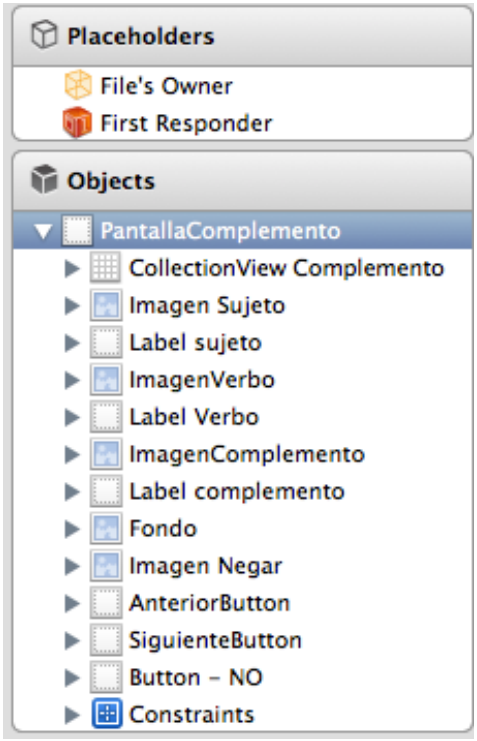


Y el comportamiento asignado en el *File's Owner* quedaría definido de la siguiente forma:

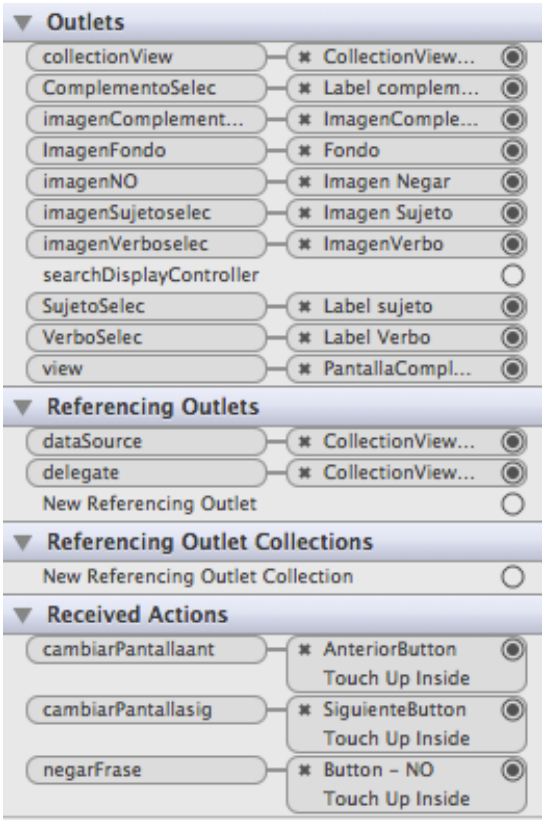


Clase PantallaComplemento

La vista de la clase PantallaComplemento quedaría compuesta por:

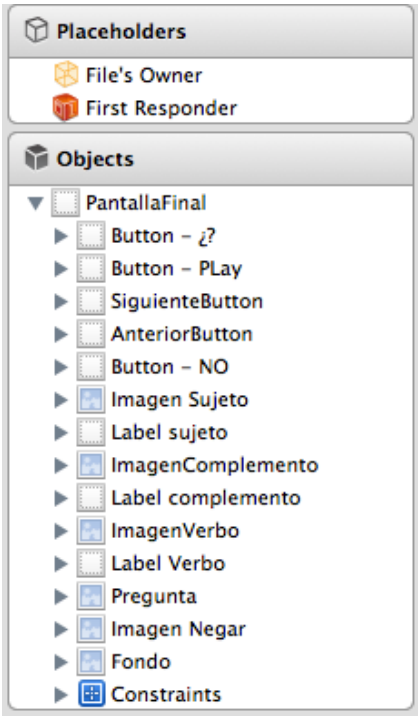


Y el comportamiento asignado en el *File's Owner* quedaría definido de la siguiente forma:

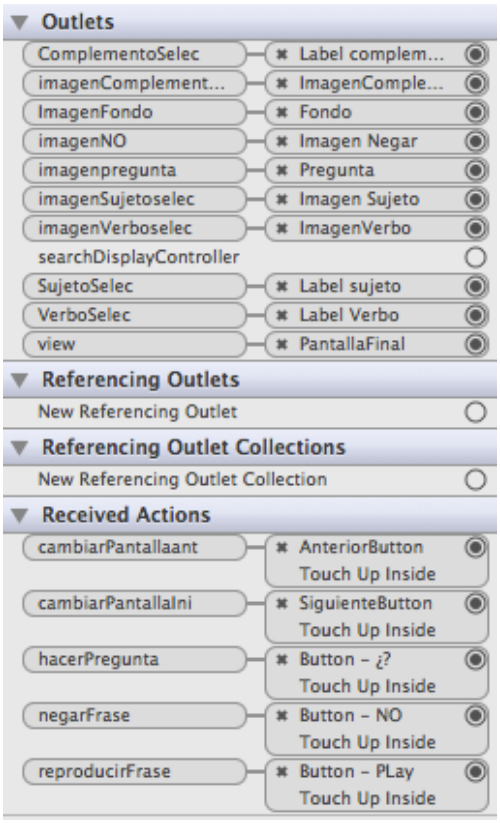


Clase PantallaFinal

La vista de la clase PantallaFinal estaría formada por los siguientes objetos:

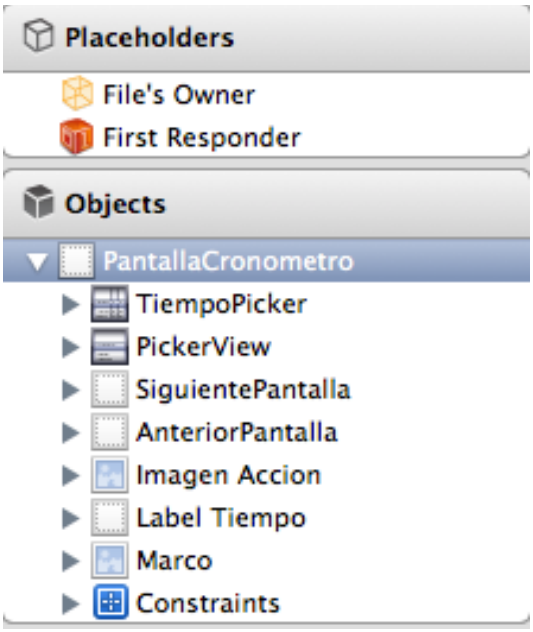


Y el comportamiento asignado en el *File's Owner* quedaría definido de la siguiente forma:

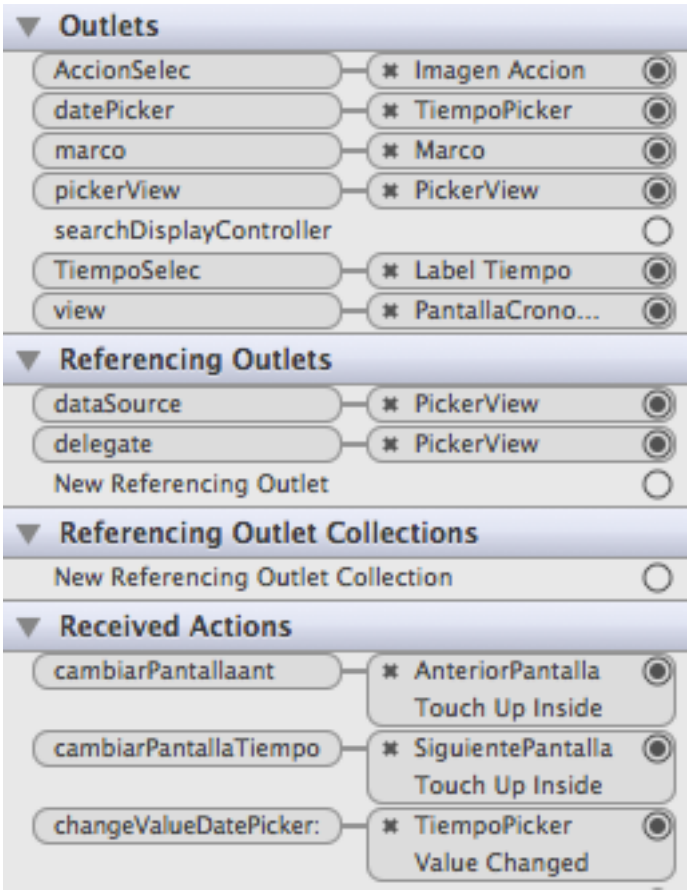


Clase PantallaCronometro

La vista de la clase PantallaCronometro estaría formada por los siguientes objetos:

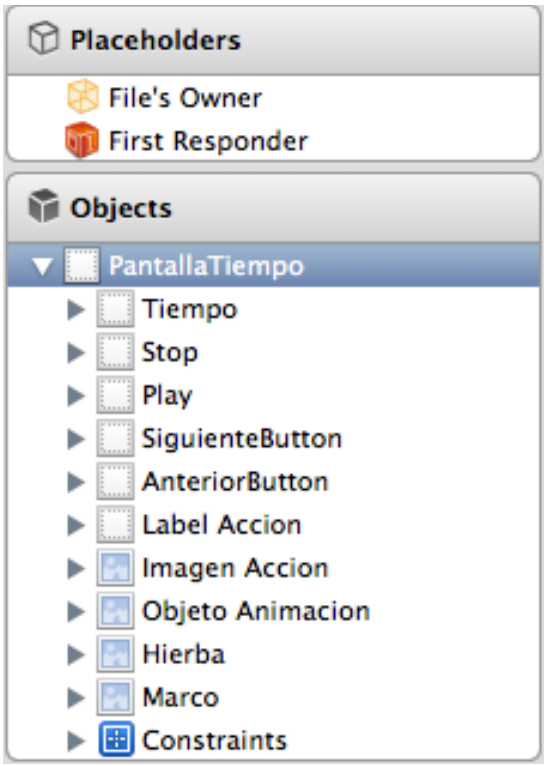


Y el comportamiento asignado en el *File's Owner* quedaría definido de la siguiente forma:

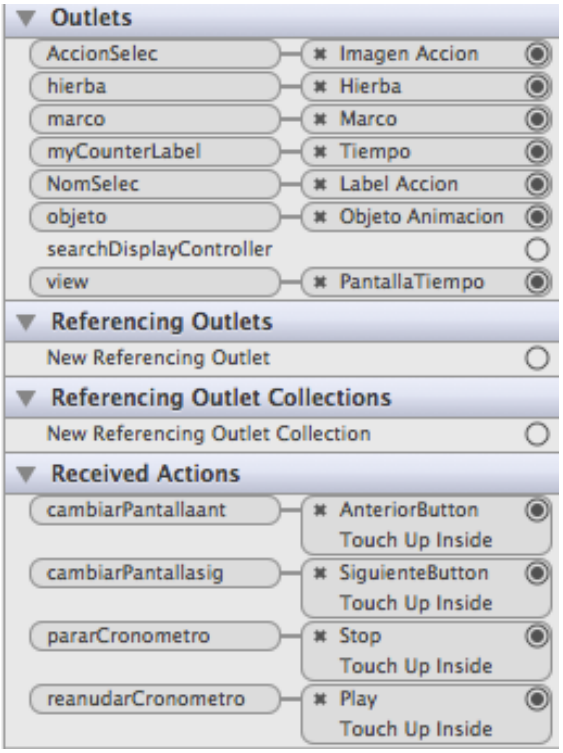


Clase PantallaTiempo

La vista de la clase PantallaTiempo estaría formada por los siguientes objetos:



Y el comportamiento asignado en el *File's Owner* quedaría definido de la siguiente forma:



3.3. DISEÑO DE LA INTERFAZ

En el apartado anterior hemos especificado todos los elementos que van a componer las distintas pantallas de las que se compone la aplicación. Ahora lo que vamos a hacer es mostrar como quedarían estas pantallas, como ha quedado la aplicación una vez introducidos estos elementos.

En el apartado de implementación se mostrará los métodos utilizados para la colocación de los distintos objetos de la vista, y para darles distintas características propias (fondo, tamaño, colores...).

Pantalla Inicial

Como ya he dicho en otros apartados será la interfaz que veremos al iniciar la aplicación, en la que decidiremos que es lo que queremos hacer.

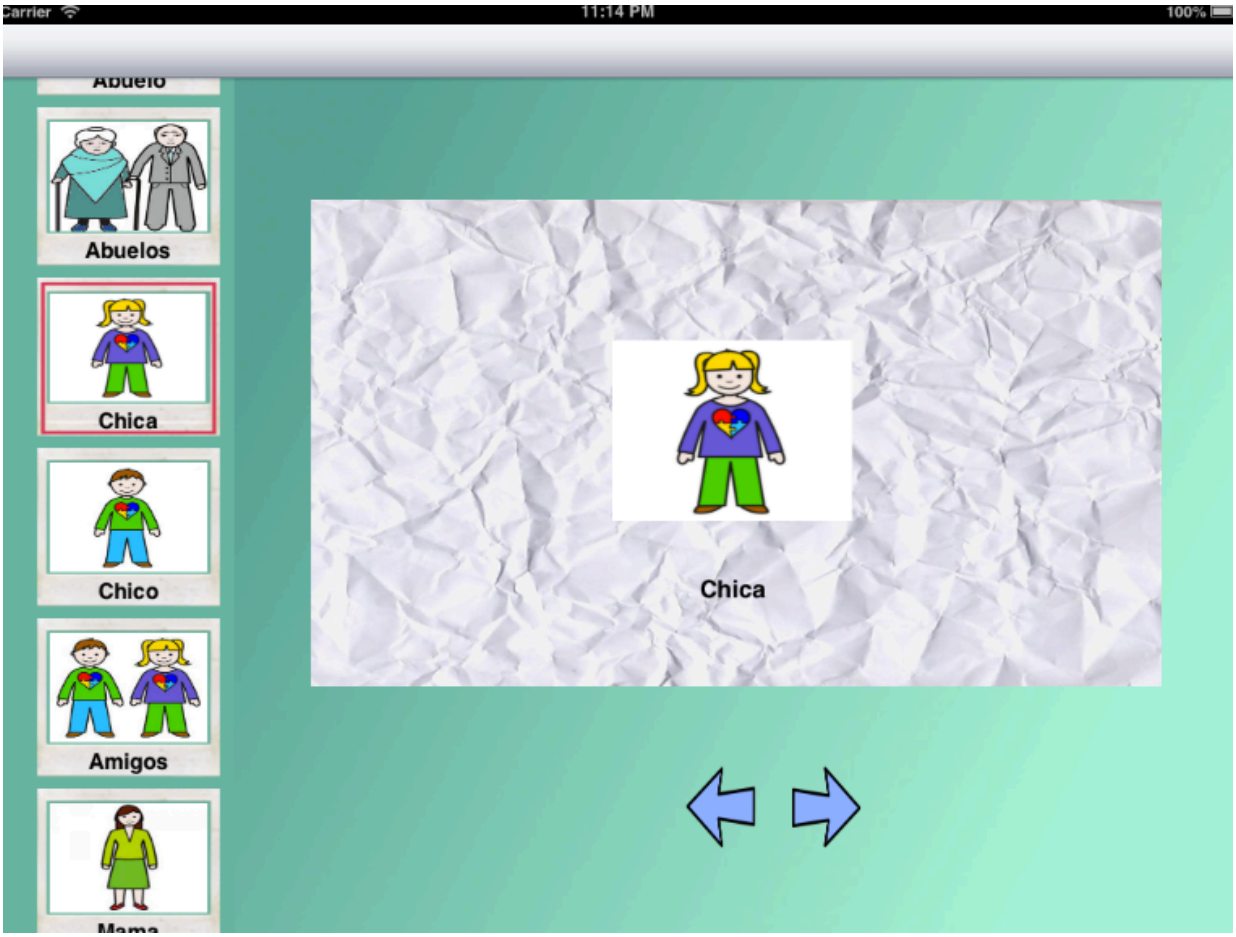
Aparecen tres botones, dos para realizar distintos tipos de frases y otro para utilizar el planificador o cronometro, que nos permitirá ver como va avanzando el tiempo.

Se ha utilizado colores poco llamativos para el fondo para que no llamen demasiado la atención del usuario y se centre más en los otros objetos que aparecen.

Pantalla Sujeto

Si por ejemplo se elige en primer lugar la opción de crear frase, botón del centro.

Aparecerá la siguiente pantalla en la que se podrá seleccionar entre unos posibles sujetos para la frase, tiene que estar al menos uno seleccionado para poder pasar a la siguiente pantalla.



La opción seleccionada es remarcada con un cuadro rojo, y aparece en el centro de la pantalla. Como podemos observar podemos ir a la siguiente pantalla o volver a atrás.

No hemos querido introducir demasiados objetos en la aplicación intentando que fuera lo más simple posible y que no distrajera al usuario, con demasiados objetos decorativos.

Pantalla Verbo

Si ahora lo que queremos es continuar con la elección de los componentes de la frase. Pasamos a la del verbo, ahora podremos seleccionar un verbo, volver a atrás o ir a la siguiente pantalla que sería la final al no tener un verbo seleccionado. En ese último caso solo podríamos escuchar el sujeto que hemos seleccionado.

Suponiendo que elegimos un verbo, como en el caso anterior quedaría resaltado el elegido, y aparecía en el centro de la pantalla junto al sujeto.

En esta pantalla nos encontramos con una nueva opción, aquí se nos permite negar la frase pulsando el botón NO.

Si lo presionamos aparecerá tachado con una cruz roja el verbo seleccionado, simulando la negación de la frase.

Si continuamos podremos elegir el complemento si ese verbo tiene alguno asignado o ir a la pantalla final directamente, si ese verbo no tiene ningún complemento asignado a él. En la parte de implementación diremos como se hacen este tipo de comprobación que son consultas a la base de datos.

Pantalla Complemento

Si nos encontramos ante un verbo con complemento, iremos a la pantalla donde podremos elegir complemento, volver a la pantalla anterior, negar la frase si no lo esta ya, en el caso de estar negada hacerla afirmativa y por último ir a la siguiente pantalla. En este último caso podremos ir a la siguiente pantalla aunque no tengamos ningún complemento seleccionado, y la frase estará compuesta por el sujeto y el verbo que tengamos previamente.

Como en las otras pantallas si algo es seleccionado quedará resaltado, y aparecerá en el centro de la pantalla junto con los demás componentes de la frase elegidos con anterioridad.

Pantalla Final

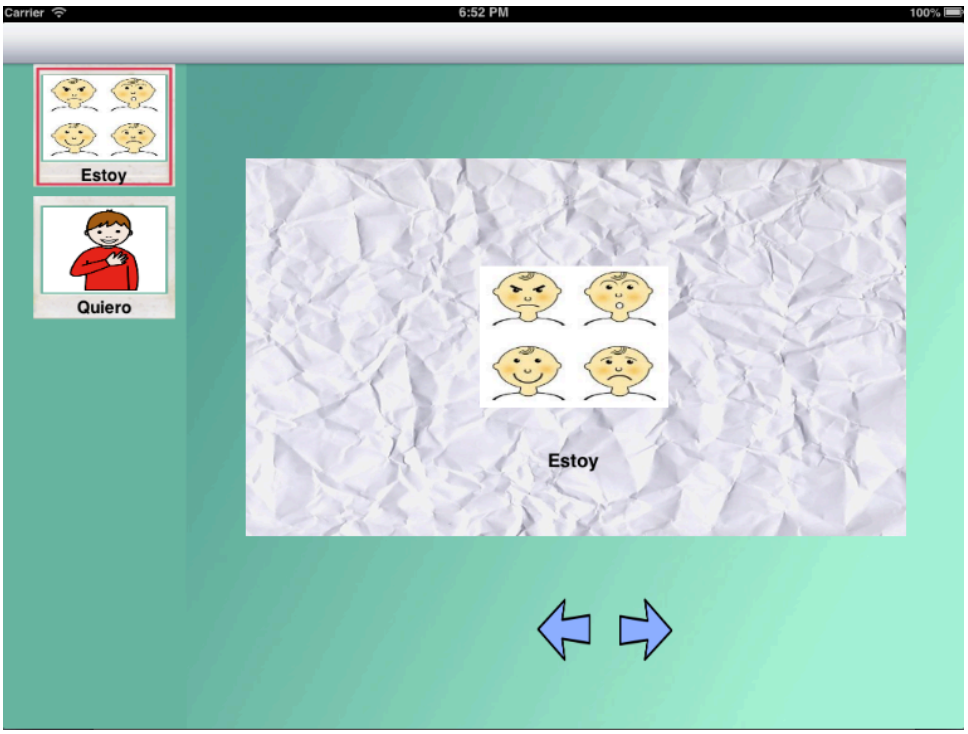
Una vez terminada la frase llegamos a la pantalla final. Aquí podemos modificar otra vez si queremos negar la frase o dejarla afirmativa, y hacer que sea pregunta. Por último podremos reproducir la frase creada.

Como en las demás pantallas podremos ir a la pantalla anterior o a la siguiente, que en este caso la siguiente será la inicial.

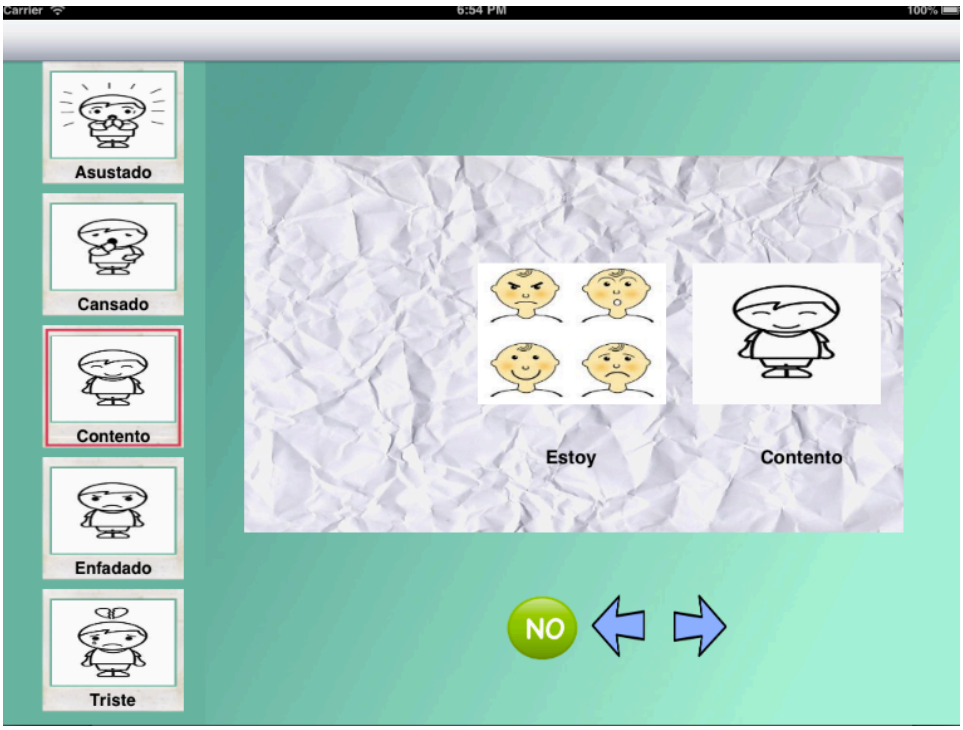
Si en vez lo que queremos es realizar una frase quiero/estoy lo único que cambia respecto a lo anterior es que en vez de elegir un sujeto tendremos que elegir entre quiero y estoy en la primera pantalla. Luego dependiendo de la elección iremos a la elección de un verbo si es Quiero o un complemento si lo que se ha elegido es estoy.

Pantalla Quiero/Estoy

Si elegimos Estoy quedará la pantalla del siguiente modo:



Después iríamos a la siguiente pantalla donde elegiríamos un estado de animo (complemento):



Pantalla Cronometro

Si lo que queremos es ir al cronometro para la realización de una acción determinada iremos a la pantalla del cronometro, desde el menú principal seleccionando el primer botón (botón del reloj).

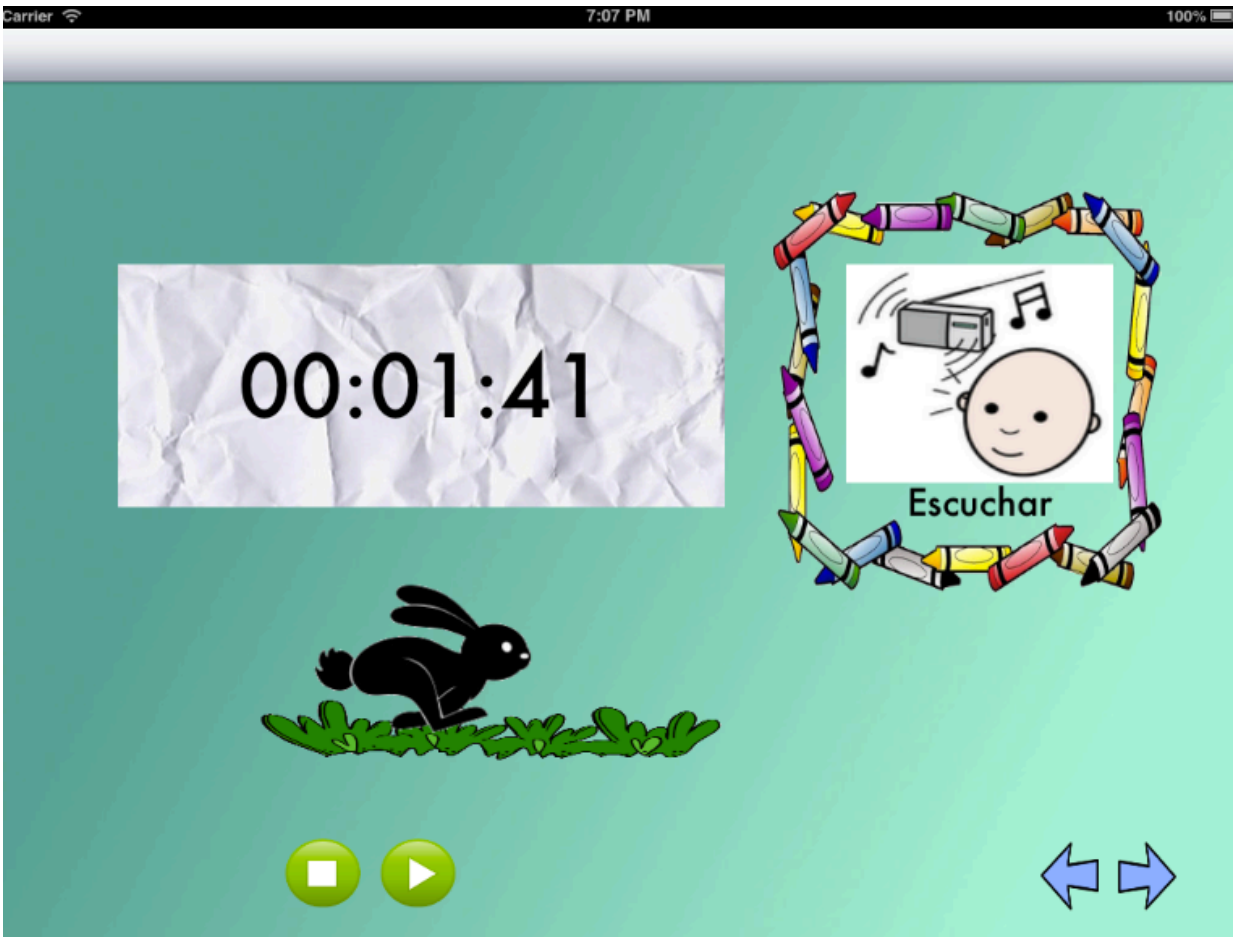
En el primer *picker* elegiremos la acción a realizar aparecen todo tipo de acciones. En el segundo, el tiempo que se va a dedicar a hacer esa determinada acción.

Para cada una de las dos elecciones aparecerá en el caso de la acción la imagen a la derecha del *picker*, y en el caso del tiempo aparecerá en una etiqueta a su derecha también el tiempo que este siendo seleccionado.

Podemos volver a la anterior pantalla, menú principal o ir a la siguiente pantalla tiempo donde comenzará la simulación del tiempo.

Pantalla Tiempo

En la última pantalla veremos como ya se ha dicho la reproducción de la animación. Aparecerá la cuenta atrás del tiempo seleccionado y el movimiento de una imagen (conejo) conforme el tiempo avanza, simulando el paso del tiempo. El conejo llegará al final cuando el tiempo termine.



Podremos volver a la pantalla anterior e ir a la siguiente pantalla como en las demás pantallas. Y además podremos parar la reproducción y volverla a reanudar.

Vemos que aparecerá la imagen de la acción con acción elegido, también veremos una etiqueta donde aparezca el cronometro.

3.4. MODELO RELACIONAL: BASE DE DATOS

A continuación veremos el paso a tablas de las tablas que necesitaremos para la creación de la base de datos de la aplicación, necesaria para acceder a los datos de los distintos componentes para formar una frase.

TABLAS DE LAS ENTIDADES

- La tabla **sujeto**:

```
CREATE TABLE "sujeto" ("sujeto" VARCHAR NOT NULL,
"tipo" VARCHAR,
"imagen" VARCHAR,
PRIMARY KEY ("sujeto"))
```

Donde la imagen será el nombre de la imagen correspondiente a esa palabra, y el campo sujeto será la palabra que será la clave primaria de la tabla de todos los sujetos que tenemos almacenados en la aplicación.

- La tabla **verbo**:

```
CREATE TABLE "verbo" ("verbo" VARCHAR NOT NULL,
"tipo" VARCHAR,
"imagen" VARCHAR,
"cdirecto" VARCHAR,
PRIMARY KEY ("verbo"),FOREIGN KEY ("cdirecto") REFERENCES verbo("verbo"))
```

Donde la imagen es el nombre de la imagen que representa a la palabra. El campo verbo es la palabra y será la clave primaria de la tabla verbo.

El campo cdirecto está para representar la relación entre la tabla con ella misma. En el caso de que nos encontremos con una frase que comience por el verbo Quiero, su complemento directo estará formado por un verbo. Por eso necesitamos este campo en la tabla de verbos, para la relación del verbo Quiero con otros verbos que serán su complemento, en la frase resultante.

- Tabla **complemento**:

```
CREATE TABLE "complemento" ("complemento" VARCHAR NOT NULL,
"tipo" VARCHAR,
"imagen" VARCHAR,
PRIMARY KEY ("complemento"))
```

Como en las anteriores tendremos el nombre de la imagen en el campo imagen, las imágenes estarán en la aplicación y accedemos por el nombre. Complemento será la clave primaria de la tabla, que será la correspondiente palabra.

TABLAS DE LAS RELACIONES

- Tabla **sujverb**, relación entre la tabla sujeto y la de verbo:

```
CREATE TABLE "sujverb" ("sujeto" VARCHAR NOT NULL,"
verbo" VARCHAR NOT NULL,
PRIMARY KEY("sujeto","verbo"),
FOREIGN KEY (sujeto) REFERENCES sujeto(sujeto),
FOREIGN KEY (verbo) REFERENCES verbo(verbo))
```

La clave primaria de esta tabla que es la relación entre dos entidades (sujeto y verbo), será la clave primaria de ambas tablas.

- Tabla **verbcom**, relación entre la tabla verbo y la de complemento:

```
CREATE TABLE "verbcom" ("verbo" VARCHAR NOT NULL,
"complemento" VARCHAR NOT NULL,
PRIMARY KEY("verbo","complemento"),
FOREIGN KEY (verbo) REFERENCES verbo(verbo),
FOREIGN KEY (complemento) REFERENCES complemento(complemento))
```

La clave primaria de esta tabla, creada de la relación de dos tablas (verbo y complemento), serán como en el caso anterior, las claves primarias de ambas tablas.

Implementación

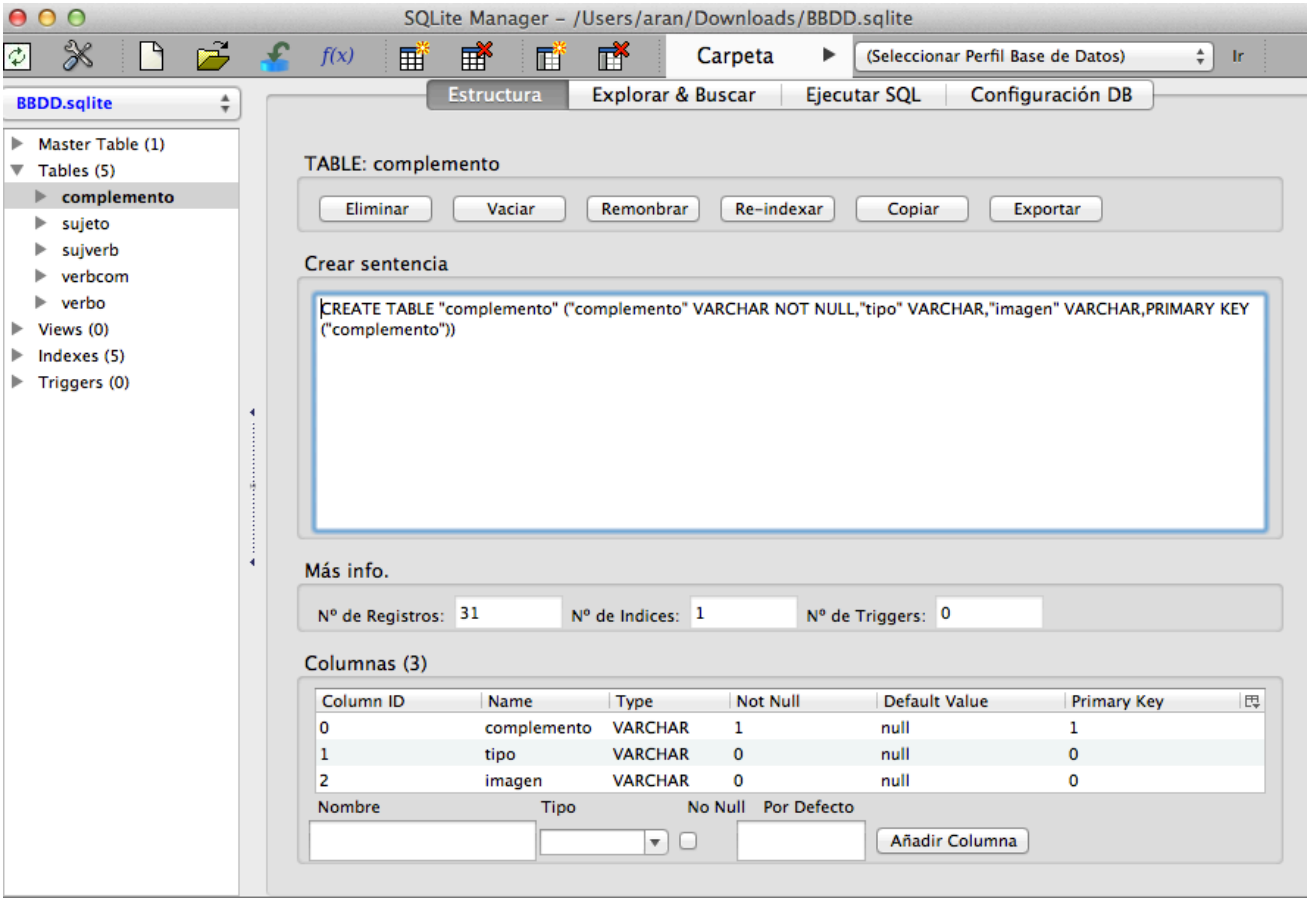
4. IMPLEMENTACIÓN

4.1. Crear Base de Datos

En primer lugar para la realización de la Base de Datos, se ha utilizado *SQLite* por simplicidad frente a otros que se podían haber utilizado como por ejemplo *Coredata*.

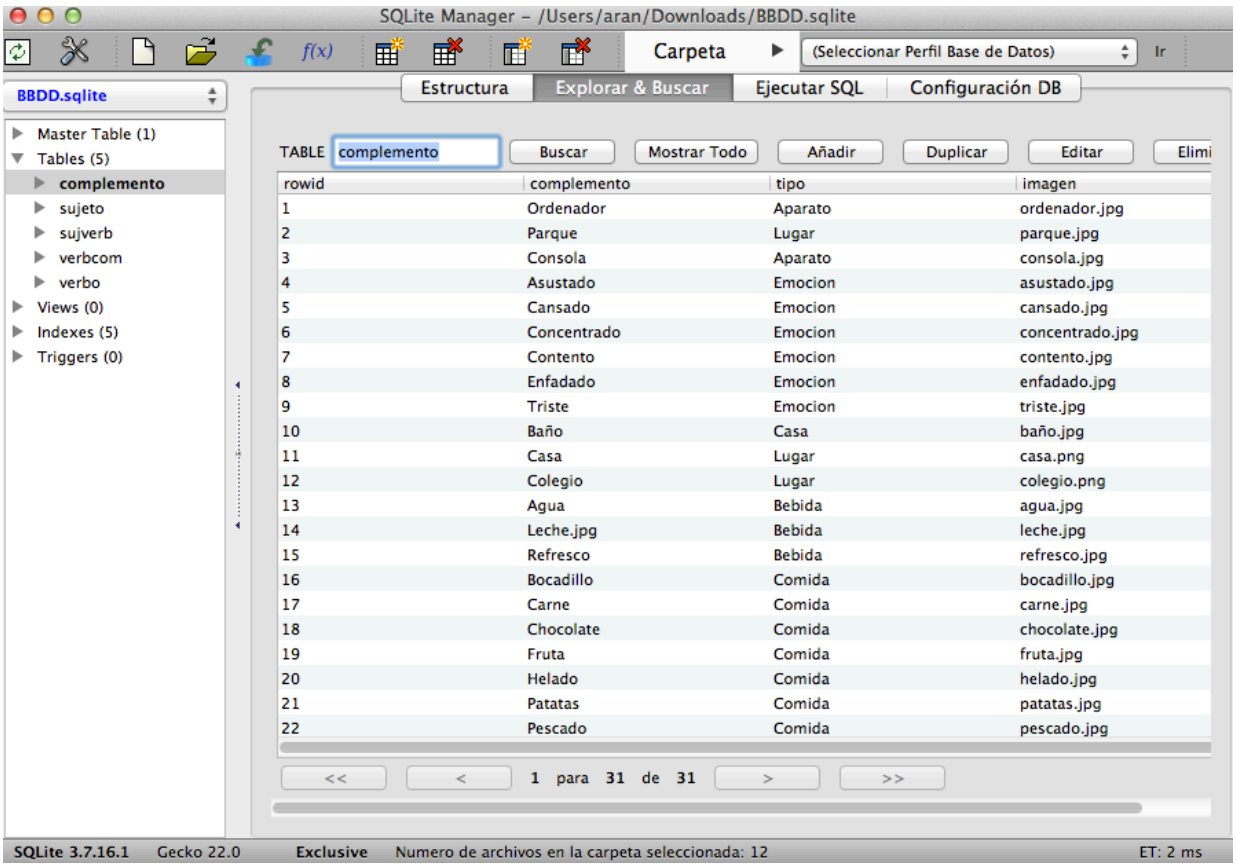
Se ha utilizado el *plugin* que proporciona Mozilla que permite la realización de bases de datos *SQLite*, *SQLite Manager*.

SQLite Manager acepta todo tipo de sentencias *SQL*, podemos ver las tablas creadas y que campos las formas.



Las tablas se ven a la derecha, si seleccionamos una de ellas como se puede observar en la imagen vemos la sentencia con la que se ha creado la tabla y debajo los campos que posee. Todo esto lo podemos ver en la pestaña de estructura.

También nos permite ver los elementos insertados en la tabla, en *Explorar & Buscar*.



Además de todo lo anterior podemos ejecutar sentencias SQL y modificar la configuración de la base de datos si fuera necesario.

Con esto podemos ya obtener la base de datos que utilizaremos para la aplicación.

Solo necesitaremos arrastras el fichero que proporciona *SQLite Manager* a la aplicación en el *Xcode*.

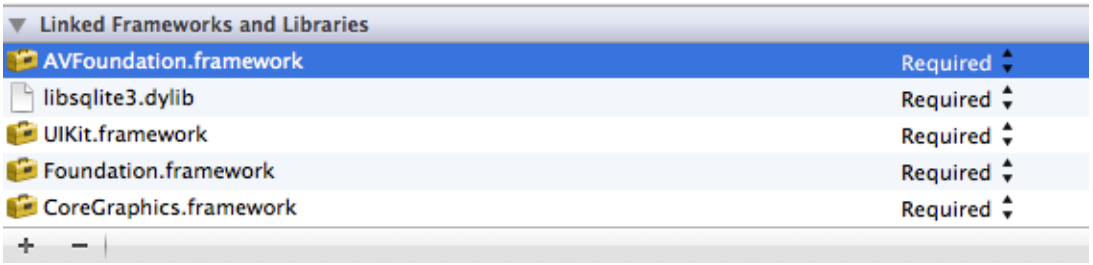
4.2. Frameworks y Librerias

Para la elaboración de la aplicación, necesitaremos añadir a los *Frameworks* y librerías unos nuevos necesarios para poder trabajar con la base de datos y para tener la posibilidad de reproducir sonidos.

Para la Base de datos necesitaremos libsqlite3.dylib, y para la realizar reproducciones *AVFoundation.framework*.

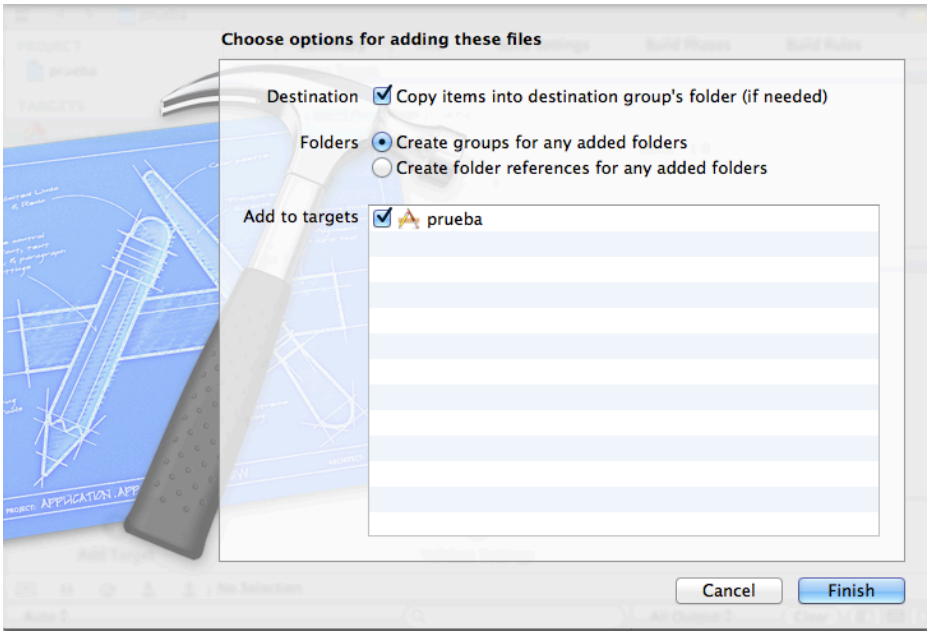
Los deberemos añadir en el Project Editor tenemos en *Sumary* se nos permite en la sección *Lincked Frameworks* and *Libraries*, añadir o eliminar *frameworks*.

A continuación, podemos ver los *Frameworks* y *libraries* de la aplicación.



4.3. Imágenes

Para añadir las imágenes al proyecto lo haremos como con la base de datos arrastrando la carpeta al proyecto en *XCode*. Al hacer esto aparece la siguiente pantalla:



Tienen que estar marcado el proyecto y la opción "*copy items into destinatio group's folder*". Para poder utilizar las imágenes en la aplicación.

4.4 Clase AppDelegate

El *Appdelegate* es el controlador de la aplicación se encarga de crear la ventana y controlador principal, que nos permitirá hacer la navegación entre las ventanas.

```
//líneas por defecto. Crea la ventana, y el controlador principal.
self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
self.viewController = [[ViewController alloc] initWithNibName:@"ViewController" bundle:nil];

// Creamos el navigation controller, poniéndole como principal suyo al view controller
UINavigationController *navController = [[UINavigationController alloc]
    initWithRootViewController:self.viewController];

// Establecemos como principal de la ventana el navigation.
self.window.rootViewController = navController;
[self.window makeKeyAndVisible];
return YES;
```

Aquí podemos ver como creamos la ventana principal de la ventana de navegación y el controlador de la navegación. La navegación de esta aplicación empezará con la clase *viewController* que será la pantalla principal al encender la principal.

También guardaremos el *path* de donde tenemos guardada la base de datos, que moveremos a una carpeta del dispositivo.

```
// Ruta para la base de datos. Estará en Library que es privada, ya que Documents se comparte
    con el usuario mediante iTunes.
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSLibraryDirectory, NSUserDomainMask, YES);
NSString *libraryDirectory = [paths objectAtIndex:0];

// Añadimos el nombre del fichero de base de datos.
self.databasePath = [libraryDirectory stringByAppendingPathComponent:@"BBDD.sqlite"];

// Cargo la base de datos
[self loadDB];
```

Y por último tenemos el método loadDB encargado de iniciar la base de datos y moverla si fuera necesario al directorio *Library*.

Hemos elegido este directorio porque al sincronizar el dispositivo con iTunes esa carpeta no se ve modificada, que es lo que queremos para poder almacenar la base de datos de la aplicación.

```
// Método para iniciar la base de datos y moverla de directorio si fuera necesario. La paso a
    Library para evitar que se muestre en la carpeta pública de iTunes.
-(void)loadDB
{
    BOOL exito;

    NSFileManager *fileManager = [NSFileManager defaultManager];

    NSError *error;

    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSLibraryDirectory, NSUserDomainMask, YES);

    NSString *libraryDirectory = [paths objectAtIndex:0];

    NSString *writableDBPath = [libraryDirectory stringByAppendingPathComponent:@"BBDD.sqlite"];

    exito = [fileManager fileExistsAtPath:writableDBPath];

    if (exito) return;

    // Si no existe en Library, la copio desde el original.
    NSString *defaultDBPath = [[[NSBundle mainBundle] resourcePath]
        stringByAppendingPathComponent:@"BBDD.sqlite"];

    exito = [fileManager copyItemAtPath:defaultDBPath toPath:writableDBPath error:&error];

    if (!exito) {
        NSAssert1(0, @"Error al cargar la base de datos, error = '%@'.", [error localizedDescription
            ]);
    }

}
```

Guardaremos el path de la base de datos (*databasePath*), para poder lanzar las sentencias SQL desde las otras clases del código de la aplicación.

También crearemos una instancia a las variables de la clase global que hemos llamado frase, porque nos ayudará a poder construirla. Aunque también se ha añadido una variable que necesitábamos para poder realizar el cronometro la otra funcionalidad de la aplicación.

```
//Para acceder a las variables de global a través de appdelegate
frase = [[Global alloc] init];
```

La frase nos permitirá saber también si la frase esta negada o si es del tipo Quiero/Estoy o es una frase normal.

Por último como ya he dicho tenemos la variable tiempo, a la que accedemos también desde frase, para no crear más variables para poder acceder a global.

4.5. Navegación

Para poder cambiar de una pantalla a otra, en el caso de la primera pantalla según que botón se seleccione iremos a una pantalla u otra.

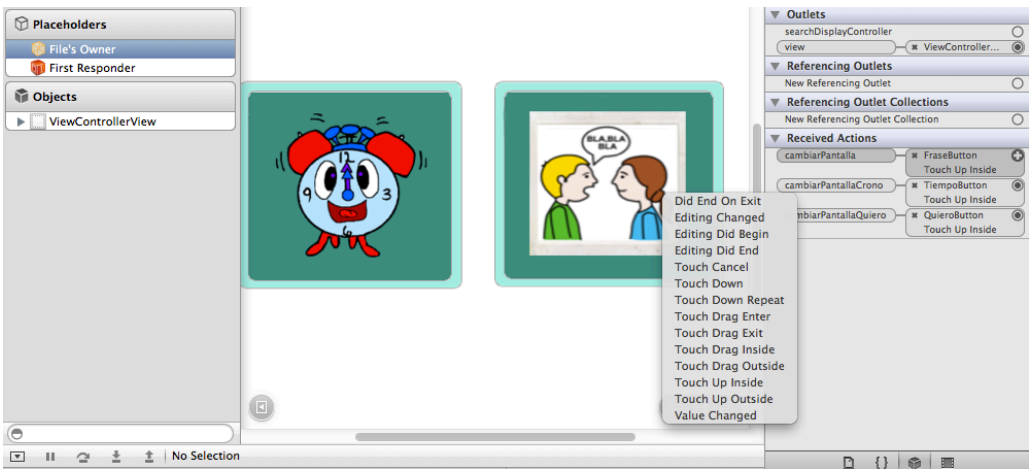
En el caso de querer formar una frase además de la navegación almacenamos que tipo de frase vamos a realizar.

Los métodos se ejecutan al realizar una acción en el botón al que ha sido asignado.

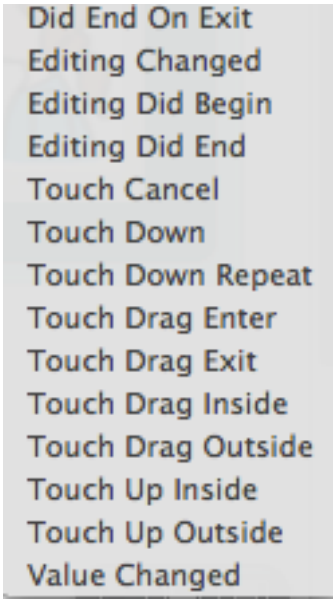
La variable botón es la que nos dice si se ha seleccionado Quiero/Estoy o Frase normal.

```
- (IBAction)cambiarPantalla
{
    //Frase completa
    appDelegate.frase.boton=@"frase";

    //Navegación cambio de pantalla
    PantallaSujeto *pantalla1 = [[PantallaSujeto alloc]initWithNibName:nil bundle:nil];
    [self.navigationController pushViewController:pantalla1 animated:YES];
}

//Cambiar a la pantalla Quiero/Estoy
-(IBAction)cambiarPantallaQuiero
{
    //Frase quiero/estoy
    appDelegate.frase.boton=@"quiero";

    //Navegación
    PantallaSujeto *pantalla = [[PantallaSujeto alloc]initWithNibName:nil bundle:nil];
    [self.navigationController pushViewController:pantalla animated:YES];
}

//Cambiar a la pantalla Cronometro
-(IBAction)cambiarPantallaCrono
{
    //NAvegación
    PantallaCronometro *pantalla = [[PantallaCronometro alloc]initWithNibName:nil bundle:nil];
    [self.navigationController pushViewController:pantalla animated:YES];
}
```

Podemos elegir entre distintas acciones que debe realizar el usuario para activar la acción.



Para poder desplegar las acciones hay que unir el método con el botón al que lo queremos asignar y se desplegará el menú que hemos visto antes, todo esto desde el File's Owner.



De todas esas opciones hemos seleccionado para todas las navegaciones que hemos utilizado en la aplicación, y también para la selección de los pictogramas "*Touch Up Inside*".

En el caso de la pantalla final cuando queramos ir a la pantalla anterior dependerá de cuál es la última pantalla en la que hemos seleccionado algo. Siempre que vayamos atrás en la aplicación borraremos el último de los datos que hemos seleccionado.

En este método haremos las comprobaciones y modificaciones necesarias para poder ir a la pantalla correcta.

```
- (IBAction)cambiarPantallaant{

    //Método para volver a atras, pantalla anterior

    if(appDelegate.frase.complemento !=nil && appDelegate.frase.verbo!=nil){
        int i= [appDelegate.frase.frase indexOfObject:
appDelegate.frase.complemento];
        [appDelegate.frase.frase removeObjectAtIndex:i];
        appDelegate.frase.complemento=nil;
        appDelegate.frase.imagenc=nil;

    PantallaComplemento *pantalla = [[PantallaComplemento
alloc]initWithNibName:nil bundle:nil];
    [self.navigationController pushViewController:pantalla animated:YES];
    }
    else if(appDelegate.frase.verbo!=nil &&
appDelegate.frase.complemento==nil){

        int i= [appDelegate.frase.frase indexOfObject:
appDelegate.frase.verbo];
        [appDelegate.frase.frase removeObjectAtIndex:i];
```

```
        appDelegate.frase.verbo=nil;
        appDelegate.frase.imagenv=nil;

        PantallaVerbo *pantalla = [[PantallaVerbo alloc]initWithNibName:nil
bundle:nil];
        [self.navigationController pushViewController:pantalla animated:YES];

    }
    else if (appDelegate.frase.verbo==nil &&
appDelegate.frase.complemento==nil){
        int i= [appDelegate.frase.frase indexOfObject:
appDelegate.frase.sujeto];
        [appDelegate.frase.frase removeObjectAtIndex:i];
        appDelegate.frase.sujeto=nil;
        appDelegate.frase.imagens=nil;

        PantallaSujeto *pantalla = [[PantallaSujeto alloc]initWithNibName:nil
bundle:nil];
        [self.navigationController pushViewController:pantalla animated:YES];

    }

}
```

4.6 Métodos clases crear frase

Hay algunos métodos que podemos encontrar en varias de las clases de la aplicación, que varían muy poco de una clase a otra, aquí vamos a explicar un poco estos métodos.

MÉTODO NEGACIÓN

En varias pantallas tenemos la opción de poder negar la frase (tachamos el verbo al negarla), lo podemos hacer en la pantalla de elegir verbo, complemento y en la pantalla final donde reproducimos la frase.

En la pantalla principal inicializamos la variable que nos dice si esta negada o no (en el *ViewController*).

```
//NEGACIÓN al principio no estará negado si le da al boton en la pantallas cambiará según el
    valor
appDelegate.frase.negado=@"NO";
```

Después en las clases correspondientes a las pantallas que hemos dicho antes podemos modificar el estado de la variable.

Para negar la frase al pulsar el botón, está el método negarFrase:

```
//BOTON NEGACIÓN
//Pasará al estado contrario según si esta negado pasará a afirmación y si no
negará la frase
//Aparece un x sobre el verbo.
-(IBAction)negarFrase{
    if(appDelegate.frase.verbo!=nil){

    //NEGACIÓN
    if ([appDelegate.frase.boton isEqual:@"quiero"]){
        if ([appDelegate.frase.negado isEqual:@"NO"]) {
            [appDelegate.frase.frase replaceObjectAtIndex: 0 withObject:
@"NO"];
            appDelegate.frase.negado=@"SI";
            if ([appDelegate.frase.sujeto isEqual:@"quiero"]){
                imagenNO = [[UIImageView alloc]
initWithFrame:CGRectMake(280,214,200,150)];
            }
            else{
                imagenNO = [[UIImageView alloc]
initWithFrame:CGRectMake(508,214,200,150)];

            }

            //Si por alguna razon no se ha seleccionado nada ponemos que sea
transparente
            if (appDelegate.frase.negado==nil){
                self.imagenNO.alpha=0;
            }
            self.imagenNO.image=[UIImage imageNamed:@"tachar.png"];
            self.imagenNO.clipsToBounds = YES;
            //self.imagenVerboselec.backgroundColor = [UIColor blackColor];
            [self.view addSubview:self.imagenNO];

        }
        else if ([appDelegate.frase.negado isEqual:@"SI"]){
            [appDelegate.frase.frase replaceObjectAtIndex: 0 withObject:
@"SI"];

            appDelegate.frase.negado=@"NO";
            self.imagenNO.alpha=0;
        }

    }
    else{
        if ([appDelegate.frase.negado isEqual:@"NO"]) {
            [appDelegate.frase.frase replaceObjectAtIndex: 1 withObject:
@"NO"];

            appDelegate.frase.negado=@"SI";

            imagenNO = [[UIImageView alloc]
initWithFrame:CGRectMake(508,214,200,150)];

            //Si por alguna razon no se ha seleccionado nada ponemos que sea
transparente
            if (appDelegate.frase.negado==nil){
                self.imagenNO.alpha=0;
            }
```

```
            self.imagenNO.image=[UIImage imageNamed:@"tachar.png"];
            self.imagenNO.clipsToBounds = YES;
            //self.imagenVerboselec.backgroundColor = [UIColor blackColor];
            [self.view addSubview:self.imagenNO];

        }
        else if ([appDelegate.frase.negado isEqual:@"SI"]){
            [appDelegate.frase.frase replaceObjectAtIndex: 1 withObject:
@"SI"];

            appDelegate.frase.negado=@"NO";

            self.imagenNO.alpha=0;
        }
    }
    }
}
```

Al negar la frase con el método anterior modificaremos la variable NO, la frase que tenemos almacenada para reproducirla en la última pantalla y mostraremos la imagen con la que tachamos el verbo.

MÉTODOS BASE DE DATOS

Para acceder a las palabras guardadas en la base de datos, tenemos el siguiente método.

```
- (void)loadCompFromDB
{
    sqlite3 *database;

    //Comprobar conexión con la BBDD
    if(!(sqlite3_open([appDelegate.databasePath UTF8String], &database) ==
SQLITE_OK)){
        NSLog(@"No se puede conectar con la BD");
    }

    // Abrimos la base de datos de la ruta indicada en el delegate
    if(sqlite3_open([appDelegate.databasePath UTF8String], &database) ==
SQLITE_OK) {

        // Podríamos seleccionar solo algunos, o todos en el orden deseado
así:
        //Sentencia a ejecutar en la BBDD
        NSString *querySQL = [NSString stringWithFormat: @"Select
c.complemento, c.tipo,c.imagen from complemento c, verbcom v where
c.complemento=v.complemento and v.verbo=\"%@\"",appDelegate.frase.verbo];
        const char *sqlStatement=[querySQL UTF8String];
        sqlite3_stmt *selectstmt=nil;

        // Lanzamos la consulta y recorremos los resultados si todo ha ido OK
        if(sqlite3_prepare_v2(database, sqlStatement, -1, &selectstmt, NULL)
== SQLITE_OK) {


            // Recorremos los resultados.
```

```
            while(sqlite3_step(selectstmt) == SQLITE_ROW) {

                // Leemos las columnas necesarias. Aunque algunos valores son
numéricos, prefiero recuperarlos en string y convertirlos luego, porque da
menos problemas.
                NSString *compdb = [NSString stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 0)];
                NSString *tipodb = [NSString stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 1)];
                NSString *imagendb = [NSString stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 2)];

                //Almacenamos los resultado en el objeto de tipo Complemento
                Complemento *auxComplemento = [[Complemento alloc]init];
                auxComplemento.complemento = compdb;
                auxComplemento.tipo=tipodb;
                auxComplemento.imagen= imagendb;

                [compArray addObject:auxComplemento];
            }

        } else {
            // Informo si ha habido algún error
            NSLog(@"ERROR en la BBDD");
        }

        // Libero la consulta
        sqlite3_finalize(selectstmt);

    }
    // Cierro la base de datos
    sqlite3_close(database);
}
```

En el método anterior accedemos al path de la base de datos, conectaremos con la base de datos y después lanzaremos la consulta correspondiente, en el caso anterior para mostrar las complementos.

Las consultas para el sujeto serían las siguientes:

```
if([appDelegate.frase.boton isEqual: @"frase"]){
            querySQL=[NSString stringWithFormat: @"SELECT * FROM sujeto"];
        }
        else {

            querySQL=[NSString stringWithFormat: @"SELECT verbo,tipo,imagen
FROM verbo WHERE verbo.verbo=\"Estoy\" or verbo.verbo=\"Quiero\" "];

        }
```

Si es una frase entera mostraremos todos los sujetos como en el primer caso, y si es quiero estoy lo que se muestra serán estas dos opciones.

Para mostrar los verbos ejecutaremos la siguiente:

```
if ([appDelegate.frase.boton isEqual: @"frase"]){
            querySQL = [NSString stringWithFormat: @"Select v.verbo,
v.tipo,v.imagen,v.cdirecto from verbo v, sujverb s where v.verbo=s.verbo and
```

```
s.sujeto=\"%@\"",appDelegate.frase.sujeto];
        }
        else {
            querySQL = [NSString stringWithFormat: @"SELECT * from verbo where
cdirecto=\"%@\"",appDelegate.frase.sujeto];

        }
```

Si estamos haciendo una frase entera mostraremos los verbos correspondientes al sujeto que se ha seleccionado previamente.

Si por el contrario estamos en una frase que comienza por quiero, lo que haremos es seleccionar los verbos que tienen quiero en el campo cdirecto.

Por último, en el caso de la clase verbo necesitaremos acceder a la base de datos para averiguar si el verbo seleccionado tiene o no complemento, y poder así al ir a la siguiente pantalla ir a la pantalla complemento o a la pantalla final según el resultado.

Obteniendo el siguiente método, que nos devolverá el número de complementos para ese verbo:

```
- (NSInteger)loadCountFromDB
{
    sqlite3 *database;

    if(!(sqlite3_open([appDelegate.databasePath UTF8String], &database) ==
SQLITE_OK)){
        NSLog(@"No se puede conectar con la BD");
    }

    // Abrimos la base de datos de la ruta indicada en el delegate
    if(sqlite3_open([appDelegate.databasePath UTF8String], &database) ==
SQLITE_OK) {

        NSLog(@" conectar con la BD");

       NSString *querySQL;

        querySQL = [NSString stringWithFormat: @"Select count(verbo) from
verbcom where verbo=\"%@\"",appDelegate.frase.verbo];

        const char *sqlStatement=[querySQL UTF8String];

        sqlite3_stmt *selectstmt=nil;

        if(sqlite3_prepare_v2(database,sqlStatement, -1, &selectstmt, NULL) ==
SQLITE_OK) {

            // Recorremos los resultados. En este caso no habrá.

            while(sqlite3_step(selectstmt) == SQLITE_ROW) {


                // Leemos las columnas necesarias. Aunque algunos valores son
numéricos, prefiero recuperarlos en string y convertirlos luego, porque da
menos problemas.
                NSString *countdb = [NSString stringWithUTF8String:(char
```

```
*)sqlite3_column_text(selectstmt, 0)];

                count = [countdb intValue];
            }

        } else {
            // Informo si ha habido algún error
            NSLog(@"Failed to open database at with error
%s",sqlite3_errmsg(database));        }

        // Libero la consulta
        sqlite3_finalize(selectstmt);
    }
    // Cierro la base de datos
    sqlite3_close(database);
    return count;
}
```

A continuación podemos ver como lanzamos el método anterior y como va a una pantalla u otra según su resultado.

```
//Método para ir a la siguiente pantalla
- (IBAction)cambiarPantallasig{
    //Accedemos al numero de complementos
    count=[self loadCountFromDB];

    //Comprobamos si para el verbo había complemento
    if (count>0){
        //Si tiene vamos a complemento
        PantallaComplemento *pantalla = [[PantallaComplemento
alloc]initWithNibName:nil bundle:nil];
        [self.navigationController pushViewController:pantalla animated:YES];
    }
    else{
        //Si no hay vamos directamente a la pantalla final
        PantallaFinal *pantalla = [[PantallaFinal alloc]initWithNibName:nil
bundle:nil];
       [self.navigationController pushViewController:pantalla animated:YES];

    }

}
```

En el count tenemos el número de complementos para el verbo que se ha seleccionado, si es cero vamos directamente a la pantalla final.

MÉTODOS COLLECTIONVIEW

En todas las pantallas cargaremos los resultados de las consulta que hemos visto en el apartado anterior contra la base de datos.

Con el siguiente método (*numberOfSections*), cargaremos el número de secciones que va a tener el *collectionview* en nuestro caso va a ser solo una sección.

```
- (NSInteger)numberOfSections:(UICollectionView *)collectionView {
    // Devuelve el número de secciones
    return 1;
}
```

Además hay que especificar el número de ítems que vamos a tener en cada sección. En nuestro caso uno por cada una de las líneas recuperadas en las consultas a la base de datos.

Aquí como ejemplo para los verbos tenemos todos los verbos almacenados en una tabla (verbosArray), por tanto tendremos un ítem por verbo en la tabla.

```
- (NSInteger)collectionView:(UICollectionView *)collectionView
numberOfItemsInSection:(NSInteger)section {
    //Devuelve el número de items por seccion del collectionView
    return [verbosArray count];
}
```

Para poder asignar a la dimensiones que va a tener hemos utilizado el siguiente método:

```
- (UIEdgeInsets)collectionView:(UICollectionView *)collectionView
layout:(UICollectionViewLayout*)collectionViewLayout
insetForSectionAtIndex:(NSInteger)section;
{
    //Tamaño para cada Sección del collectionView
    return UIEdgeInsetsMake(10, 10, 0, 10);
}
```

Para cargar las celdas tenemos el método *cellForItemAtIndexPath* crearemos una celda para cada elemento de la tabla, le asignaremos a cada elemento de la celda la información que hemos obtenido.

```
- (CollectionCell *)collectionView:(UICollectionView *)collectionView
cellForItemAtIndexPath:(NSIndexPath *)indexPath {

    //identificador de Celda
    static NSString *CellIdentifier=@"Cell";

    //Creamos la celda
    CollectionCell *cell = [collectionView

dequeueReusableCellWithReuseIdentifier:CellIdentifier
                              forIndexPath:indexPath];

    //contenido de la tabla de los verbos
```

```
    Verbo *auxVerbo = [verbosArray objectAtIndex:indexPath.row];

    //Ponemos el background de la celda
    cell.backgroundView = [[UIImageView alloc] initWithImage:[UIImage
imageNamed:@"photo-frame.png"]];
    //asignamos la imagen correspondiente al verbo
    cell.imagen.image=[UIImage imageNamed:[auxVerbo imagen]];
    //Asignamos la palabra a la label
    cell.label.text=[auxVerbo verbo];
    //Fondo para imagen seleccionada
    cell.selectedBackgroundView = [[UIImageView alloc] initWithImage:[UIImage
imageNamed:@"photo-frame-selected.png"]];

    //Devuelve la celda
    return cell;
}
```

Le asignamos por un lado la imagen, la etiqueta con el nombre y la imagen de fondo. Y además le asignamos otra imagen que servirá para mostrar si una celda esta seleccionada.

Para saber cuál es el pictograma que esta seleccionado, necesitamos *didSelectItemAtIndexPath*:

```
- (void)collectionView:(UICollectionView *)collectionView
didSelectItemAtIndexPath:(NSIndexPath *)indexPath {

    // Recuperamos la posición del Array igual a la fila seleccionada.
    Verbo *auxVerbo = [verbosArray objectAtIndex:indexPath.row];

    //seleccionamos la imagen y el verbo para mostrarlo
    //Cambiamos la imagen a mostrar
    [self.imagenVerboselec performSelectorOnMainThread:@selector(setImage:)
withObject: [UIImage imageNamed:[auxVerbo imagen]] waitUntilDone:YES];
    //Cambiamos el texto
    VerboSelec.text=[auxVerbo verbo];


    if (appDelegate.frase.verbo==nil){
        [appDelegate.frase.frase addObject:auxVerbo.verbo];
    }
    else{

        int i= [appDelegate.frase.frase indexOfObject:
appDelegate.frase.verbo];
        [appDelegate.frase.frase replaceObjectAtIndex: i withObject:
auxVerbo.verbo];
    }
    //Se guarda para mostrar en la siguiente pantalla el Verbo seleccionado
    appDelegate.frase.verbo=auxVerbo.verbo;
    appDelegate.frase.imagenv=auxVerbo.imagen;
    NSString *string=[appDelegate.frase.frase componentsJoinedByString:@"/"];

}
```

Además de cambiar el estado de la celda ha el de una celda seleccionada. Lo que hacemos es almacenar que palabra hemos elegido para la frase o en el

caso de que ya tuviéramos una elegida lo que haremos modificar la selección anterior.

VIEWDIDLOAD AND VIEWDIDAPPEAR

El *viewdidload* lo que hace es cargar los datos de la vista al cargar la aplicación y *viewdidAppear* lo que hace es cargarlos cuando la vista es visible (también puede ser cargados porque se han hecho modificaciones).

En cada una de las clases tendremos que cargar unos datos, conforme vayamos pasando a otras pantallas tendremos que cargar lo que hemos seleccionado en la pantallas anteriores.

En todas ellas tendremos que cargar el fondo de pantalla:

```
  //Imagen Fondo
    UIColor *color = [[UIColor alloc] initWithPatternImage:[UIImage
imageNamed:@"degradadoverde.jpg"]];
    self.view.backgroundColor = color;
```

Que es una imagen convertido en *UIColor* para poder establecerlo como *background* de la vista.

En las pantallas donde tengamos el *collectionView* tendremos también que cargar el fondo de este:

```
    //Imagen Fondo collectionView
    UIColor *colorc = [[UIColor alloc] initWithPatternImage:[UIImage
imageNamed:@"colecverde.jpg"]];
    self.collectionView.backgroundColor=colorc;
```

También modificaremos la barra de navegación, ya que la navegación se hará por los botones. Eliminaremos la flecha que aparece arriba en la barra de navegación que pone back.

```
  //Para ocultar flecha atras de navegación, ya hay boton
    [self.navigationItem setHidesBackButton:YES];
```

En las pantallas en las que tenemos collectionView, tendremos además que llamar al método que obtiene la información de las palabras que aparecerán en la base de datos.

También le asignaremos el asignaremos el identificador a las celdas para cargar los datos en el método *cellForItemAtIndexPath:*

```
     //Array de los verbos para mostrar en el collectionView
    verbosArray = [[NSMutableArray alloc]init];
    //Lanzar conexión y consulta en la BBDD
    [self loadVerbosFromDB];

    //Asignamos a la CollectionView el identificador para cada Celda
```

```
    //El tipo de Celda-> CollectionCell
    [self.collectionView registerClass:[CollectionCell class]
forCellWithReuseIdentifier:@"Cell"
     ];
    //Actulizar el CollectionView con los datos de la BBDD
    [collectionView reloadData];
```

Para todas la pantallas asignaremos un fondo en donde se colocarán todas las imágenes con sus etiquetas (palabra) de los pictogramas que hemos seleccionado, ya sea de esa en esa pantalla o en las anteriores.

Para cargar los datos de pantallas anteriores, ponemos como ejemplo lo que haríamos en el caso de querer cargar los datos de un verbo.

Para cargar la imagen, tendríamos que dar la posición donde la vamos a colocar se la asignamos con el método "*initWithFrasem:CGRectMake*", que lo que hace es al inicio (*initWithFrame*), asignarle las coordenadas y dimensión de la imagen (*CGRect*).

```
     imagenVerboselec = [[UIImageView alloc]
     initWithFrame:CGRectMake(508,214,200,150)];
    self.imagenVerboselec.clipsToBounds = YES;
```

Una vez le hemos asignado las características que queremos para la imagen añadimos la subvista, a la vista en la que estamos:

```
   [self.view addSubview:self.imagenVerboselec];
```

Como con imagen para la etiqueta (*Label*), le tendremos que asignar las coordenadas y dimensiones con el método que hemos visto antes.

```
    VerboSelec = [[UILabel alloc] initWithFrame:CGRectMake(580,370,100,100)];
```

También le asignaremos otro tipo de característica como el color, tamaño o fondo.

```
    //Color de la label
    VerboSelec.textColor = [UIColor blackColor];
    VerboSelec.backgroundColor=[UIColor clearColor];
    //Tipo letra
    self.VerboSelec.font = [UIFont boldSystemFontOfSize:20.0];
```

Por último como el caso de la imagen añadiremos la subvista a la vista en la que estamos:

```
    [self.view addSubview:self.VerboSelec];
```

Esto lo haremos para cada elemento que hemos seleccionado y queremos que se muestre al pasar a esa pantalla.

Como ya hemos dicho en el *viewdidappear* actualizaremos los datos de la vista en la que nos encontremos, actualizaremos los datos a cargar de la base de datos para asegurarnos de que mostramos los correctos.

```
- (void)viewDidAppear:(BOOL)animated
{

    appDelegate = (AppDelegate*)[[UIApplication sharedApplication]delegate];
    verbosArray = [[NSMutableArray alloc]init];
    [self loadVerbosFromDB];
    [self loadCountFromDB];

    [collectionView reloadData];
}
```

MÉTODO PREGUNTA

Para poder hacer que la frase se convierta en una pregunta lo que haremos será mostrar el símbolo de interrogación, esto lo haremos en la pantalla final teniendo en cuenta si se había pulsado ya o no.

```
-(IBAction)hacerPregunta{
    if ([appDelegate.frase.pregunta isEqual:@"NO"]) {
        appDelegate.frase.pregunta=@"SI";

        imagenpregunta = [[UIImageView alloc]
initWithFrame:CGRectMake(836,214,200,150)];

        //Si por alguna razón no se ha seleccionado nada ponemos que
sea transparente
        if (appDelegate.frase.pregunta==nil){
            self.imagenpregunta.alpha=0;
        }
        self.imagenpregunta.image=[UIImage imageNamed:@"pregunta.png"];
        self.imagenpregunta.clipsToBounds = YES;
        //self.imagenVerboselec.backgroundColor = [UIColor blackColor];
        [self.view addSubview:self.imagenpregunta];

    }
    else if ([appDelegate.frase.pregunta isEqual:@"SI"]){
        appDelegate.frase.pregunta=@"NO";
        self.imagenpregunta.alpha=0;
    }
}
```

Como para todas las imágenes podemos ver en el método, como añadimos todas las características necesarias para mostrar la imagen o para ponerla trasparente (haciendo que el valor de *alpha* sea 0).

CREACIÓN Y REPRODUCCIÓN DE LA FRASE

Para la creación de la frase utilizaremos la frase que hemos ido almacenando, pero además la copiaremos para tener la posibilidad de poder modificarla en la pantalla final (negación, interrogación).

```
-(NSString *)crearFrase{

    NSMutableArray *array;
    array=[NSMutableArray new];
    //No puedo copiar directamente porque se modifican los objetos de los 2
arrays y solo quiero que se modifiquen en uno para dar la posibilidad que una
vez reproducido una vez pueda ser seleccionado luego la negación.
    int i;
    int count= [appDelegate.frase.frase count];
    for (i=0;i<count;i++){
        NSString *objeto=[appDelegate.frase.frase objectAtIndex:i];
        [array addObject:objeto];
    }

    if ([appDelegate.frase.negado isEqual:@"NO"]) {
        if ([appDelegate.frase.boton isEqual:@"quiero"]) {
            [array removeObjectAtIndex:0];
        }
        else{
            [array removeObjectAtIndex:1];

        }
    }
    NSString *string=[array componentsJoinedByString:@" "];

    if([appDelegate.frase.pregunta isEqual:@"SI"]){
        string=[NSString stringWithFormat:@"¿ %@ ?",string];
    }

    return string;
}
```

Después lo único que tenemos que hacer es llamar al método anterior desde el que vamos a ver a continuación con el cuál ya podremos reproducirla:

```
-(IBAction)reproducirFrase{

    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *path = [documentsDirectory
stringByAppendingPathComponent:@"file.mp3"];

    NSString *text=[self crearFrase];

    NSString *urlString = [NSString
stringWithFormat:@"http://www.translate.google.com/translate_tts?tl=es&q=%@",t
ext];
    NSURL *url = [NSURL URLWithString:[urlString
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
```

```
    NSMutableURLRequest* request = [[NSMutableURLRequest alloc]
initWithURL:url];
    [request setValue:@"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:2.0.1)
Gecko/20100101 Firefox/4.0.1" forHTTPHeaderField:@"User-Agent"];
    NSURLResponse* response = nil;
    NSError* error = nil;
    NSData* data = [NSURLConnection sendSynchronousRequest:request
                                         returningResponse:&response
                                                     error:&error];
    [data writeToFile:path atomically:YES];

    NSError        *err;

    if ([[NSFileManager defaultManager] fileExistsAtPath:path])
    {
        player = [[AVAudioPlayer alloc] initWithContentsOfURL:
                  [NSURL fileURLWithPath:path] error:&err];
        player.volume = 0.4f;
        [player prepareToPlay];
        [player setNumberOfLoops:0];
        [player play];
    }

}
```

Al no poseer ningún tipo de API en castellano que se pudiera utilizar para poder hacer el cambio de texto a voz, y las apis disponibles para *Objective C* no estaban en castellano que era lo que se requería para nuestra aplicación.

Se buscó la solución que podemos ver en el método anterior, que utiliza *google translate* para poder tener acceso a las palabras que necesitamos y poder reproducir la frase al completo.

Como ya hemos dicho al principio cuando hemos hablado de los *frameworks* y librerías que íbamos a utilizar, tenemos el *framework* "*AVFoundation.framework*" que nos va a permitir esa reproducción.

Aquí podemos ver como lo utilizamos:

```
    player = [[AVAudioPlayer alloc] initWithContentsOfURL:
              [NSURL fileURLWithPath:path] error:&err];
    player.volume = 0.4f;
    [player prepareToPlay];
    [player setNumberOfLoops:0];
    [player play];
```

4.7. Clase Cronometro

En la clase cronometro aunque necesitaremos métodos como el de cargar los datos de los verbos que tenemos en la base de datos, para poder elegir entre las distintas acciones que hay. Tenemos otros métodos que necesitaremos como:

PICKER VIEW

Como en el caso del *collectionView* tiene una serie de métodos que utilizaremos:

En el método "*numberOfComponentsInPickerView*" asignaremos el número de componentes que habrá en este caso un componente o sección:

```
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView;
{
    return 1;
}
```

Por otro lado asignaremos el número de filas que tendrá cada componente, en nuestro caso será igual al número de verbos de la base de datos:

```
- (NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent:(NSInteger)component;
{
    return [verbosArray count];
}
```

La palabra que mostraremos en cada una de las filas del *pickerview*:

```
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
forComponent:(NSInteger)component;
{

    Verbo *auxVerbo=[verbosArray objectAtIndex:row];

    return auxVerbo.verbo;

}
```

La fila que se ha quedado seleccionada, la guardaremos porque la mostraremos en esta pantalla, pero también en la siguiente pantalla aparecerá la imagen junto con la palabra que representa:

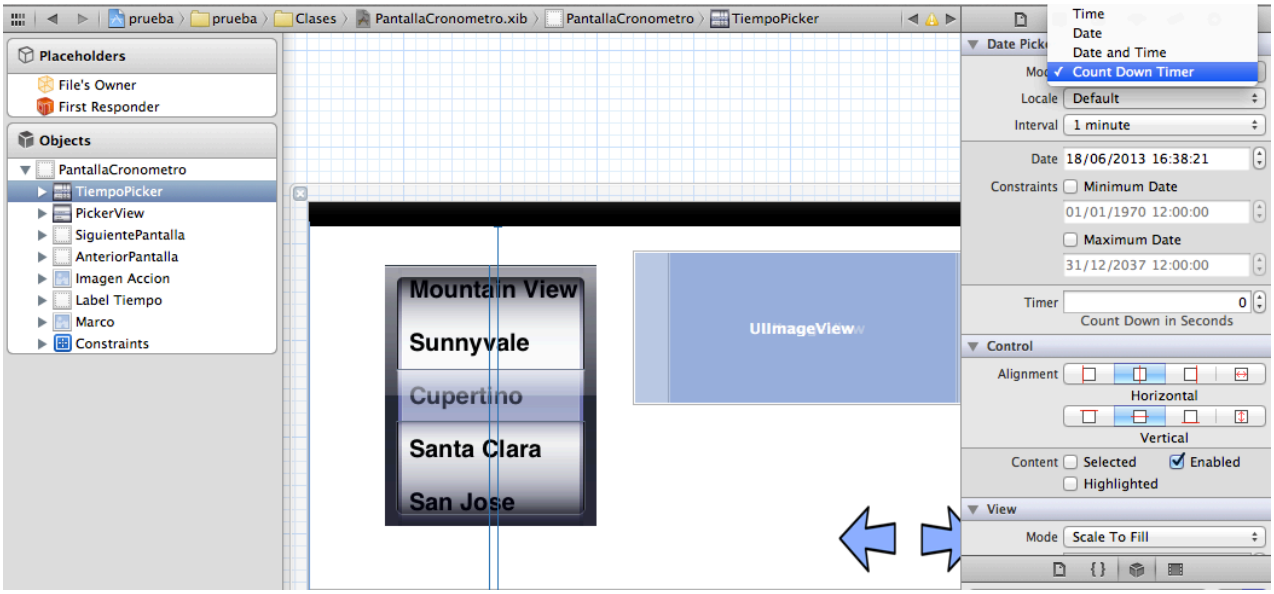
```
- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
inComponent:(NSInteger)component
{

    Verbo *auxVerbo=[verbosArray objectAtIndex:row];

    //Utilizo la del verbo para poder mostrar la imagen en la pantalla del
tiempo
    appDelegate.frase.verbo=auxVerbo.verbo;
    appDelegate.frase.imagenv=auxVerbo.imagen;
```

```
    //IMAGEN
    AccionSelec = [[UIImageView alloc]
initWithFrame:CGRectMake(450,80,300,200)];
    AccionSelec.image=[UIImage imageNamed: auxVerbo.imagen];
    [self.view addSubview:self.AccionSelec];
}
```

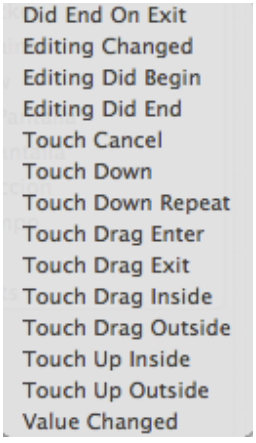
DATE PICKER

En el caso de *Date Picker* utilizaremos aquel en el que aparecen horas y minutos, hay más opciones como fechas (mes, dia...) y el de fecha y hora.

Podemos modificarlo según queramos el que nosotros utilizamos será el de tipo *count down timer*. Como podemos ver en la imagen en las opciones del Datepicker podemos modificarlo:



Además le asignaremos la correspondiente acción para obtener el valor que hemos elegido:



En este caso usaremos “Value changed” que coge el último valor seleccionado.

Guardaremos el tiempo seleccionado que luego utilizaremos en la pantalla tiempo donde se simulará como va avanzando, lo guardamos en segundos para simplificar ya para su posterior utilización:

```
-(IBAction)changeValueDatePicker:(UIDatePicker *)sender{
    appDelegate.frase.tiempo=0;

    NSDateFormatter *dateformatter = [[NSDateFormatter alloc] init];
    [dateformatter setDateFormat:@"HH:mm:ss"];

     NSDateComponents *time = [[NSCalendar currentCalendar]
                               components:NSHourCalendarUnit |
NSMinuteCalendarUnit
                               fromDate:[datePicker date]];
    appDelegate.frase.tiempo=[time hour]*60*60+[time minute]*60;

    //Mostrar el tiempo en la label
    self.TiempoSelec.text=[dateformatter stringFromDate:[datePicker date]];
}
```

4.8. Clase Tiempo

Como ya hemos visto necesitaremos cargar una serie de imágenes para esta clase y lo haremos en el *viewdidload*.

Cargaremos el fondo como en todas las pantallas, además de un fondo para las etiquetas, el marco donde mostraremos la acción seleccionada en la pantalla anterior y las imágenes necesarias para la reproducción (hierva y conejo).

```
- (void)viewDidLoad {

    //Imagen Fondo
    UIColor *color = [[UIColor alloc] initWithPatternImage:[UIImage
imageNamed:@"degradadoverde.jpg"]];
    self.view.backgroundColor = color;
    [self.navigationItem setHidesBackButton:YES];

    appDelegate = (AppDelegate*)[[UIApplication sharedApplication]delegate];


    //Fondo etiqueta
    myCounterLabel=[[UILabel alloc]
initWithFrame:CGRectMake(100,150,500,200)];
    myCounterLabel.textColor=[UIColor blackColor];
    myCounterLabel.textAlignment = NSTextAlignmentCenter;
    [myCounterLabel setFont:[UIFont fontWithName:@"Futura" size:70]];
    UIColor *coloreti = [[UIColor alloc] initWithPatternImage:[UIImage
imageNamed:@"Papelarrugado.jpg"]];
    myCounterLabel.backgroundColor = coloreti;
    myCounterLabel.text=[NSString stringWithFormat:@"00:00:00"];

    [self.view addSubview:self.myCounterLabel];

    [super viewDidLoad];
    secondsLeft=0;
    secondsLeft = appDelegate.frase.tiempo;

    [self countdownTimer];
```

```
    STOP=TRUE;

    //HIERBA
    hierba = [[UIImageView alloc] initWithFrame:CGRectMake(200,500,400,60)];
    hierba.image=[UIImage imageNamed:@"hierba.png"];
    [self.view addSubview:self.hierba];

    //PARA EL OBJETO QUE SE MUEVE
    objeto = [[UIImageView alloc] initWithFrame:CGRectMake(200,400,200,150)];
    objeto.backgroundColor=[UIColor clearColor];
    objeto.image=[UIImage imageNamed:@"conejo.png"];

    [self.view addSubview:self.objeto];

    //MARCO
    marco= [[UIImageView alloc] initWithFrame:CGRectMake(640,90,320,330)];
    marco.image=[UIImage imageNamed: @"marcopinturas.png"];
    [self.view addSubview:self.marco];



    //IMAGEN acción seleccionada
    AccionSelec = [[UIImageView alloc]
initWithFrame:CGRectMake(700,150,220,180)];
    AccionSelec.image=[UIImage imageNamed: appDelegate.frase.imagenv];
    [self.view addSubview:self.AccionSelec];
    //Etiqueta Accion
    NomSelec=[[UILabel alloc] initWithFrame:CGRectMake(710,320,200,50)];
    NomSelec.textColor=[UIColor blackColor];
    NomSelec.textAlignment = NSTextAlignmentCenter;
    [NomSelec setFont:[UIFont fontWithName:@"Futura" size:30]];
    NomSelec.backgroundColor=[UIColor clearColor];
    NomSelec.text=appDelegate.frase.verbo;
    [self.view addSubview:self.NomSelec];

}
```

Una vez hemos cargado todo lo que necesitamos para esta pantalla y hemos llamado a los métodos para poder mostrar la animación, el sonido y el avance del cronometro.

Inicializamos en cronometro y le asignaremos en tiempo que se había seleccionado.

```
-(void)countdownTimer{

    hours = minutes = seconds = 0;

    timer = [NSTimer scheduledTimerWithTimeInterval:1.0f target:self
selector:@selector(updateCounter:) userInfo:nil repeats:YES];
}
```

Conforme avance el cronometro iremos actualizando los elementos de la pantalla, como el movimiento del conejo y el tiempo restante (etiqueta con el tiempo):

```
- (void)updateCounter:(NSTimer *)timer {
```

```
    if (STOP==FALSE){
        //MOVER
        posicionHorizontal = objeto.frame.origin.x;
        int recorrido=800-(posicionHorizontal+objeto.frame.size.width);

        int avance=recorrido/appDelegate.frase.tiempo;
        if (posicionHorizontal<800) {
            derecha=posicionHorizontal+avance;
        }
        CGRect rectanguloOrigen = objeto.frame;
        rectanguloOrigen.origin.x = derecha;
        objeto.frame = rectanguloOrigen;

        //ETIQUETA CON EL TIEMPO RESTANTE
        if(secondsLeft > 0 ){

            secondsLeft =secondsLeft -1;
            hours = secondsLeft / 3600;
            minutes = (secondsLeft % 3600) / 60;
            seconds = (secondsLeft %3600) % 60;

            myCounterLabel.text = [NSString
stringWithFormat:@"%02d:%02d:%02d", hours, minutes, seconds];
        }
        else{
            //Cuando entro aqui ha terminado el tiempo
            //Paramos el cronometro y la animación
            STOP=TRUE;

            //Reproducir sonido al terminar el tiempo
            NSString *soundFilePath = [[NSBundle mainBundle]
pathForResource:@"alarma"

ofType:@"mp3"];
            NSURL *soundFileURL = [NSURL fileURLWithPath:soundFilePath];
            player = [[AVAudioPlayer alloc] initWithContentsOfURL:soundFileURL
error:nil];
            [player play];
        }
    }
}
```

Como ya hemos dicho iremos avanzando el cronometro, en este caso haremos la cuenta atrás hasta llegar a cero. Cuándo lleguemos al final del tiempo lo que haremos es emitir un sonido de alarma para notificar que ya ha acabado.

Por último, podremos parar y volver a reanudar la cuenta atrás con dos botones que modificarán una variable:

```
-(IBAction)pararCronometro{
    //Preta el boton de deterner cronometro
    STOP=TRUE;
}

-(IBAction)reanudarCronometro{
    //Boton de reanudar
    if (secondsLeft!=0){
        STOP=FALSE;
    }
}
```

4.9. En resumen

Con todo lo anterior tendríamos los elementos más importantes de la implementación de la aplicación.

Lo más importante que utilizamos prácticamente todas las clases creadas, es la navegación siempre se nos permite la posibilidad de ir a la siguiente pantalla y a la anterior. La utilización de la base de datos para obtener información que mostramos por pantalla, y la actualización de la información que vamos utilizando y manipulando en cada una de las pantallas.

Además por supuesto de la reproducción de sonidos y la animación para ayudar al usuario en la compresión del paso del tiempo.

Para la utilización de algunas de las clases que utilizamos como ya he explicado, se requiere de la utilización de ciertos métodos que son modificables pero necesarios si queremos usarlas.

Pruebas

5. PRUEBAS

Al utilizarse para la aplicación la base datos hemos realizado diferentes pruebas, para comprobar que los datos mostrados y los que teníamos almacenados eran los mismos.

Además de probar el buen funcionamiento en todo lo referente a la navegación y actualización de variables, sobre todo las variables necesarias para formar la frase.

Para comprobar que se ejecuta el código se puede utilizar:

```
NSLog(@"");
```

Que nos permite mostrar información por el terminal de la aplicación o por ejemplo el siguiente mensaje:

```
NSLog(@"Failed to open database at with error
%s",sqlite3_errmsg(database));
```

Que nos muestra que error se ha producido en la base de datos al obtener los datos.

A parte de probar todas las posibilidades que pueden ejecutarse desde el simulador que nos proporciona IOS, y *XCode*.

Teniendo en cuenta que el funcionamiento debe ser sencillo porque va dirigido a personas con dificultades en el aprendizaje entre otras.

Conclusiones y Líneas Futuras

6. CONCLUSIONES Y LÍNEAS FUTURAS

Al final hemos obtenido una aplicación para iPad que cumple con la funcionalidad que se nos pedía. Que va ayudar a personas, en nuestro caso sobre todo niños con problemas en el habla o dificultades en el aprendizaje.

Con este proyecto ha servido para adquirir nuevos conocimientos asociados a los dispositivos IOS. Aprender a programar con el lenguaje *Objective C* y utilizar *XCode* para que nos ayude en la programación.

Objective C es un lenguaje orientado a objetos, creado como un superconjunto de C. Además como hemos visto al realizar una aplicación de Apple hemos tenido que utilizar en patrón de diseño Modelo Vista Controlador que utiliza para crear sus aplicaciones.

También se han adquirido conocimientos en cuanto al análisis y diseño, de la aplicación que nos ayuda a llevar a cabo el proyecto, y a tomar las decisiones que nos van surgiendo a lo largo de él.

Para la realización de la base de datos se ha utilizado también *SQLite* que aunque es similar a lo SQL, nos a permitido aprender a trabajar en conjunto con ambas lenguajes para poder utilizar la base de datos desde *Objective C*.

Y por último a buscar todo tipo de errores en el código he incluso en el simulador de IOS, utilizando todo tipo de pruebas que nos han permitido cerciorarnos de que la aplicación hacia como debía todas sus funciones.

En conclusión este proyecto, aparte de que nos permite ayudar al usuario en su aprendizaje y adaptación a la sociedad, permite que aprendamos como trabajar con otros lenguajes de programación nuevos.

Y llevar a cabo un proyecto completo, buscando la información necesaria para poder aprender en un primer momento el nuevo lenguaje de programación utilizado y luego la documentación necesaria para llevarlo a cabo.

Siempre como ya he dicho teniendo en cuenta que debía ser una interfaz sencilla y simple, que ayude a este tipo de personas (sobre todo niños), y que no les distraiga de el objetivo final.

Trabajar con otro tipo de dispositivos, en este caso *tablets* aunque podría haber sido para móviles (iPhone), ya que no habría muchos cambios solo en lo que se refiere a ajustarse al tamaño de la imagen.

Se podrían añadir modificaciones como hacer que fuera compatible con otros dispositivos Apple como es el iPhone, o incluso hacer que la aplicación fuera asequible a un mayor número de usuarios si la hiciéramos para *Android* lo que llevaría mucho más trabajo.

En cuanto a funcionalidad se podría permitir modificar los pictogramas (imágenes que aparecen), o añadir nuevos.

Podríamos crear una nueva api con los sonidos para que podríamos utilizar la aplicación offline, y permitir su uso en cualquier lugar sin depender de la conexión a internet.

Y por supuesto se podrían añadir nuevas funcionalidad que ayuden al aprendizaje y conocimiento de las palabras, como podrían ser juegos relacionados con el vocabulario.

Bibliografía

7. BIBLIOGRAFÍA

Para obtener las bases necesarias para el proyecto ha sido necesaria la realización de diversos tutoriales.

Para la elaboración de la memoria se ha consultado el libro:

- Introducción a la Ingeniería del Software. Alonso Amo, Fernando; Martínez Normand, Loïc; Segovia Pérez, Francisco Javier.

Para la realización del proyecto se han consultado diversas páginas web, además de la información proporcionada por Apple para *developers* como:

- http://stackoverflow.com/
- http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/Introduction/Introduction.html
- http://developer.apple.com/library/ios/#documentation/general/conceptual/DevPedia-CocoaCore/MVC.html
- https://developer.apple.com/devcenter/ios/index.action
- https://itunes.apple.com/us/app/xcode/id422352214
- https://addons.mozilla.org/es-es/firefox/addon/sqlite-manager/
- http://www.apprendemos.com/tutoriales/ios/sqlite-manager-apps
- http://www.apprendemos.com/tutoriales/ios/sqlite-base-de-datos-en-iphone
- http://www.dimetecnologia.com/
- www.apprendemos.com/tutoriales/ios/sqlite-base-de-datos-en-iphone

También se han consultado tutoriales de youtube como:

- http://www.youtube.com/watch?v=xh0eXOD9rg8

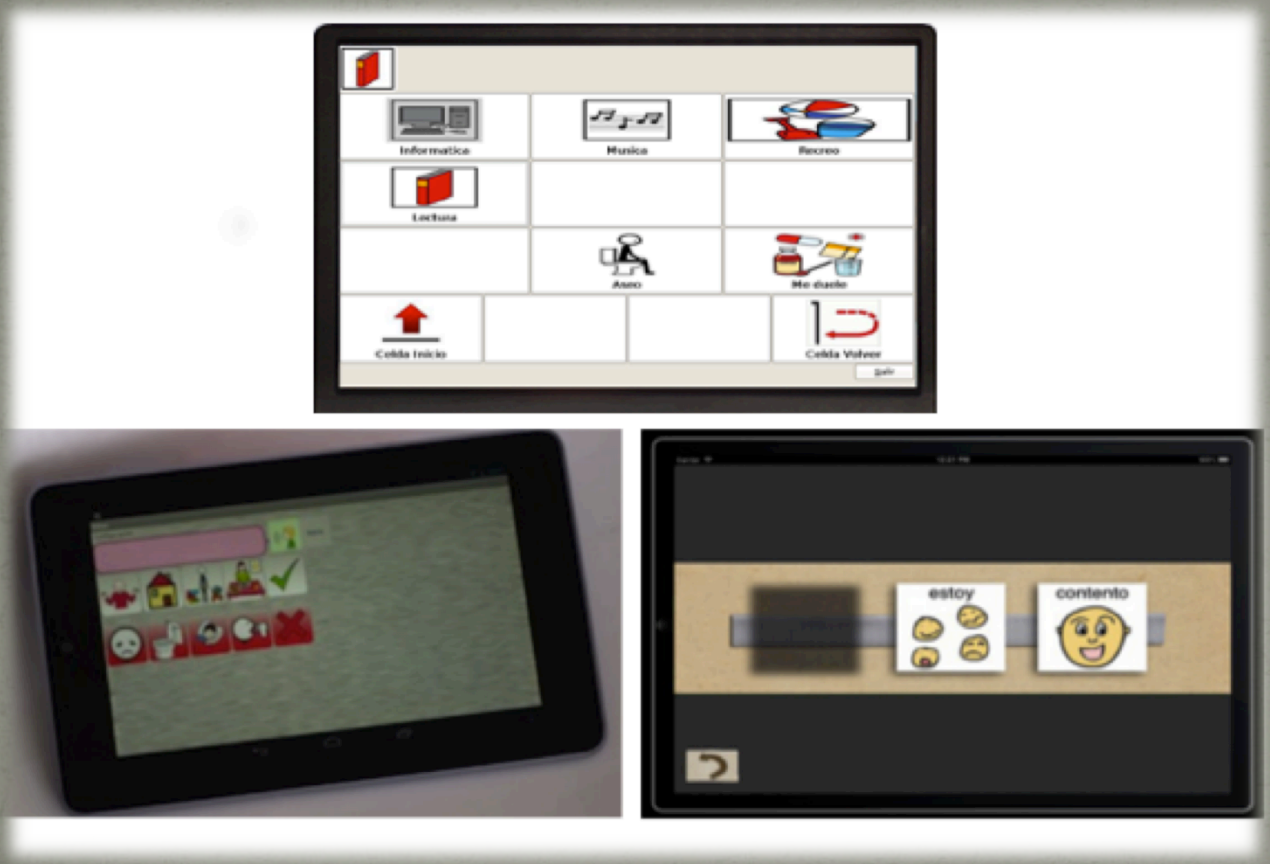
Aplicación de comunicación para personas con dificultades en el habla

Arantxa Mateo Bayo

Principales Objetivos

- Ayudar a las personas con discapacidades físicas o psíquicas, que tienen dificultades en el habla.
- Aprender a realizar un proyecto completo, en concreto una aplicación para dispositivos IOS.

Estado del arte

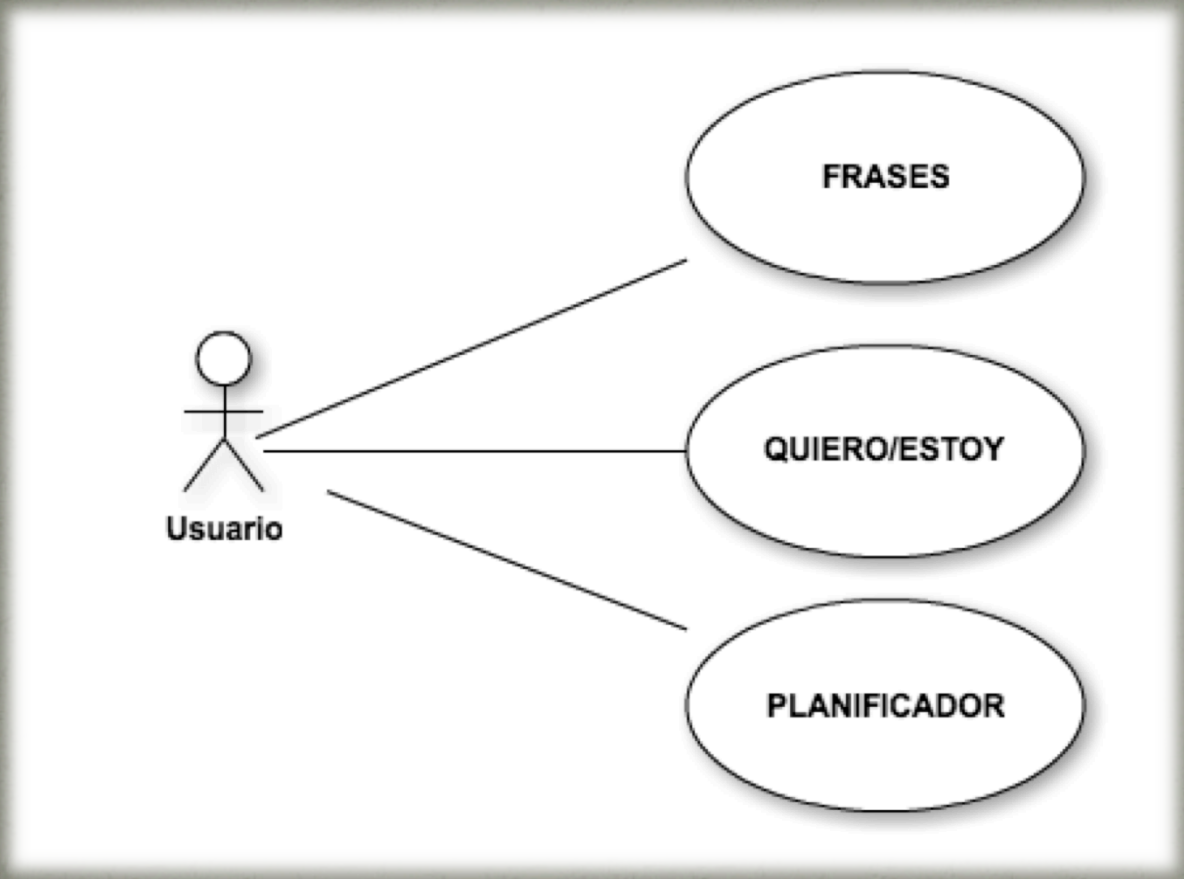


- *SC@UT*
- *Hotuba*
- *DIME*

Solución propuesta

- Aplicación IOS para iPad
- Ayudar a comunicarse y planificar el tiempo
- Creación de frases
- Cronómetro
- Utilizaremos pictogramas, sonidos y animaciones
- Proceso Unificado

Análisis



Casos de Uso

- Creación de frases
  - Quiero/Estoy
  - Frases completas
- Planificar tiempo para actividad

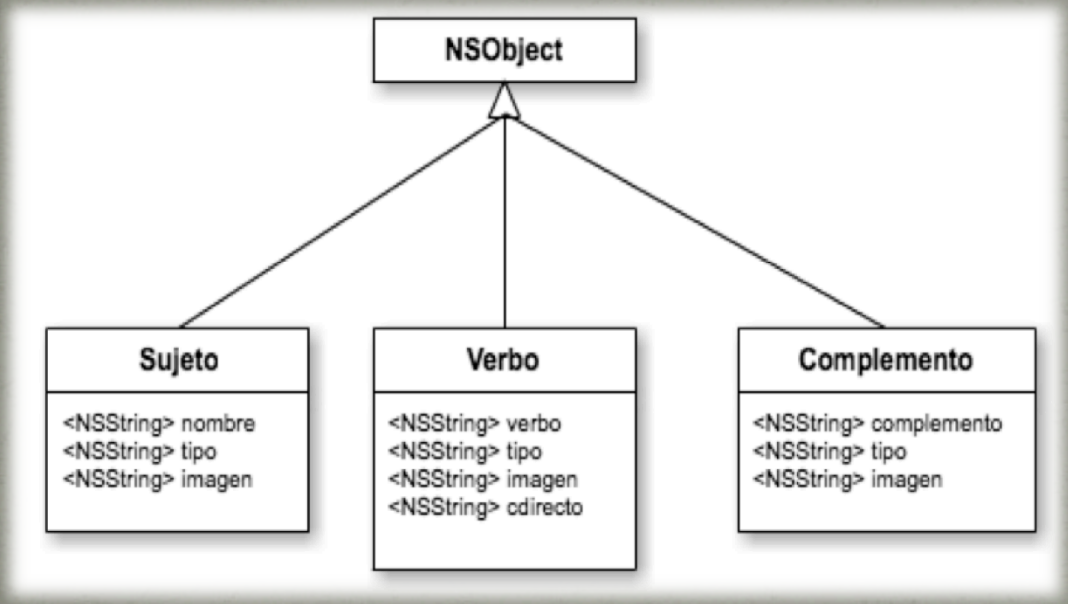
Diseño

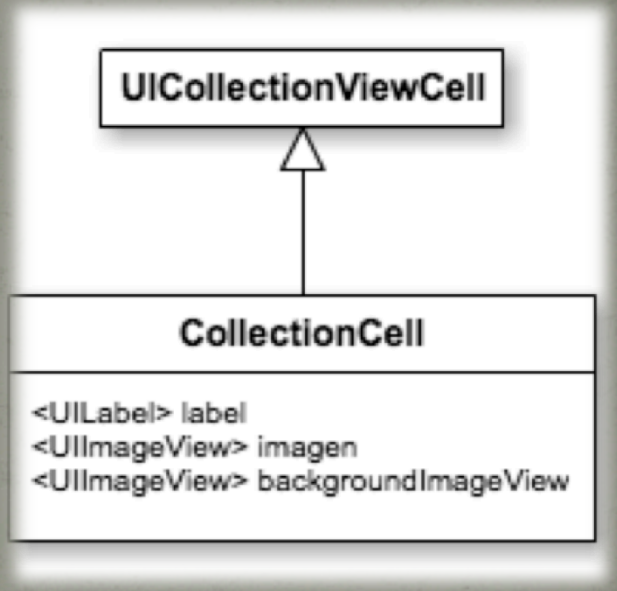
MODELO VISTA CONTROLADOR

- Modelo: representación de los datos obtenidos de la Base de datos.
- Controlador: responde a las acciones del usuario.
- Vista: interfaz gráfica.

Diseño

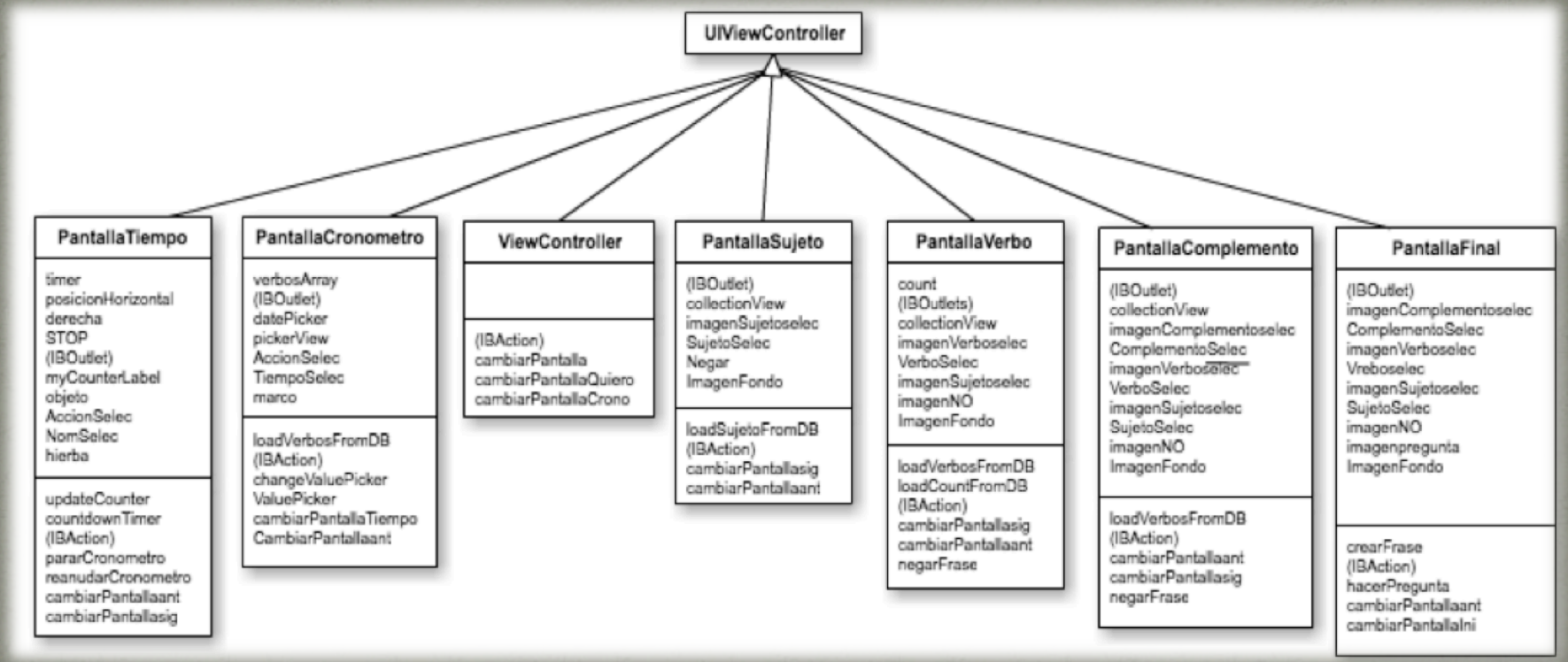
Clases de modelo





Diseño

Clases controlador

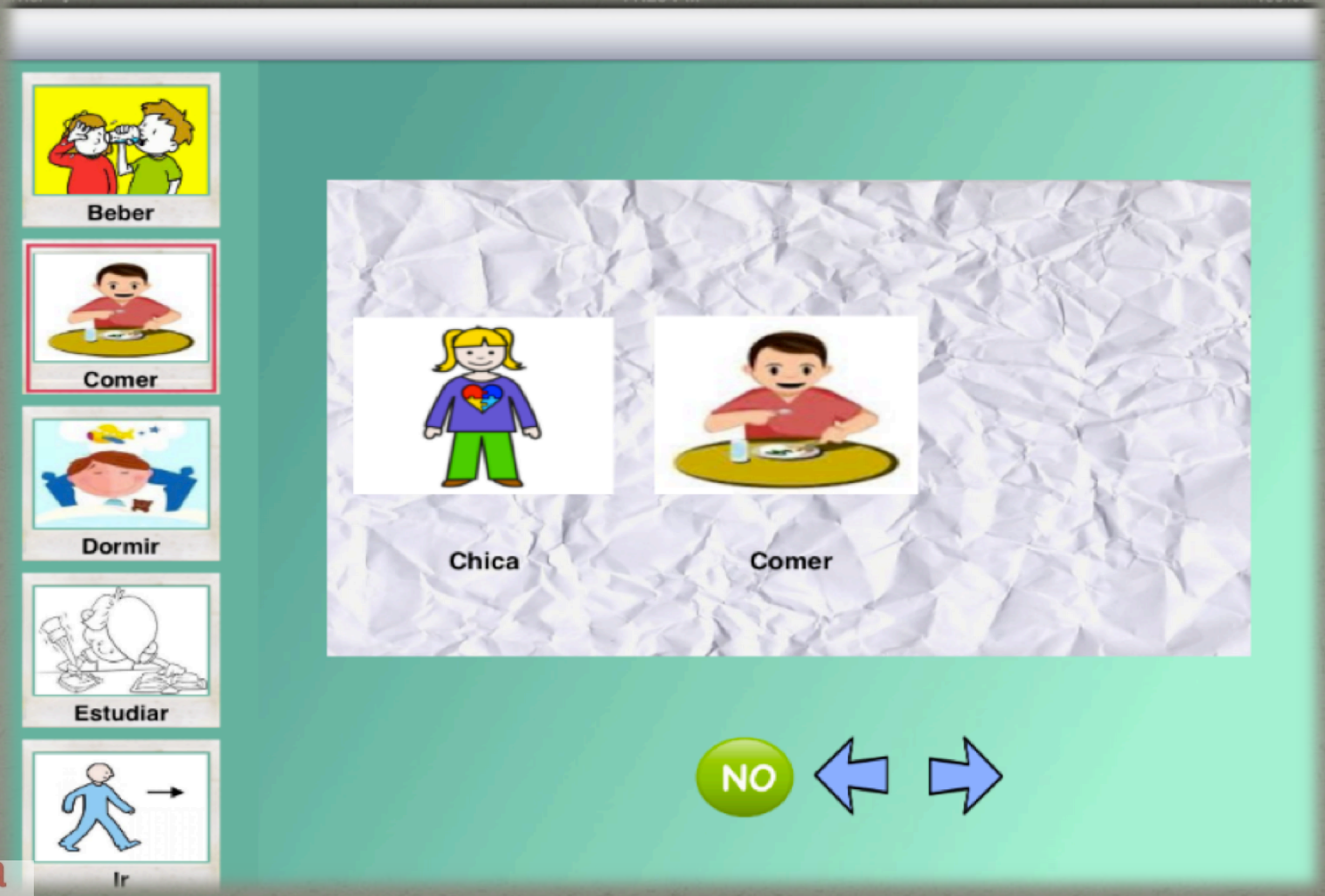


Características

Además de lo anteriormente dicho la aplicación cumplirá unas características de diseño:

- Interfaz sencilla de utilizar
- Intuitiva
- Sin distracciones
- Imágenes identificables con lo que representan

Interfaz Gráfica



Interfaz Gráfica

00:01:41

Implementación

- Base de datos SQLite
- AppDelegate
- Librerías
  - libsqlite3.dylib
  - *AVFoundation.framework*
- Problemas conversión Texto a Voz

Conclusiones

- Crear una aplicación para personas con dificultades para comunicarse
- Aprender a desarrollar una aplicación IOS
- Adquirir nuevos conocimientos

Líneas Futuras

- Añadir la posibilidad de que el usuario pueda modificar e introducir nuevos pictogramas.
- Crear una nueva API de sonidos más clara
- Introducir otras funcionalidades que ayuden en el aprendizaje de las palabras