

ALLMYLYRICS

Raquel Chamorro Giganto

**Desarrollo de
aplicaciones iOS**

Máster MIMO

**Desarrollo de
la aplicación**

ÍNDICE

1. Introducción.....	2
2. Objetivos	2
3. Herramientas y recursos.....	3
4. Descripción funcional de la aplicación	4
5. Descripción técnica de la aplicación	8
6. Análisis crítico de la aplicación	17
7. Bibliografía.....	18

La aplicación “AllMyLyrics”

1. Introducción

En términos generales, AllMyLyrics es una aplicación móvil iOS para iPhone que proporciona un motor de búsqueda de letras de canciones además de permitir almacenarlas y visualizarlas en el dispositivo o añadirlas manualmente.

2. Objetivos

Esta pequeña aplicación tiene como objetivo que su público pueda encontrar o almacenar fácilmente letras de canciones para obtener más información sobre la canción, entender la letra si está en un idioma que el usuario desconoce, aprender la canción o complementar la experiencia de escuchar música.

Para ello la aplicación proporciona:

- Un motor de búsqueda que permite hacer búsquedas concretas de canciones en una base de datos externa a la aplicación con tan solo introducir el título y los/as artistas de la canción.
- La introducción manual de letras de canciones si estas no se encuentran con el buscador externo.
- El almacenamiento local de las canciones buscadas o introducidas manualmente para tenerlas siempre disponibles sin necesidad de conexión a internet.
- Una lista donde se podrán visualizar las canciones almacenadas ordenadas alfabéticamente para su fácil búsqueda.
- La opción de borrar de una manera simple las canciones que el usuario haya almacenado en la aplicación.
- Una interfaz accesible y fácil de usar que permita realizar todo lo mencionado anteriormente.

3. Herramientas y recursos

La aplicación ha sido realizada en un ordenador MAC con el entorno de desarrollo propio de macOS **Xcode** versión 12.4 en Swift 5.

Para realizar esta aplicación he utilizado varias herramientas, todas ellos gratuitas, que se pueden obtener fácilmente.

Para la llamada a la API se ha usado **Alamofire**, una librería para realizar peticiones HTTP. En este caso solo ha sido necesaria una vez puesto que solo se utiliza la llamada a la API en el buscador de canciones.

La API usada ha sido “**lyrics.ovh**”, un proyecto pequeño de código abierto gracias al cual ha sido posible encontrar una API de este tipo y de forma gratuita. A pesar de ser simple y tener pocas opciones para hacer requests, provee a la aplicación lo necesario para obtener el objetivo principal: conseguir letras de canciones.

Otra librería usada ha sido **DesignSystem** para cambiar la fuente y los iconos que se utilizan en los botones y dar un aspecto más cuidado a la aplicación.

Las dos librerías que se han utilizado han sido añadidas al proyecto con **CocoaPods**, un administrador de dependencias para aplicaciones hechas con Swift programada con Ruby.

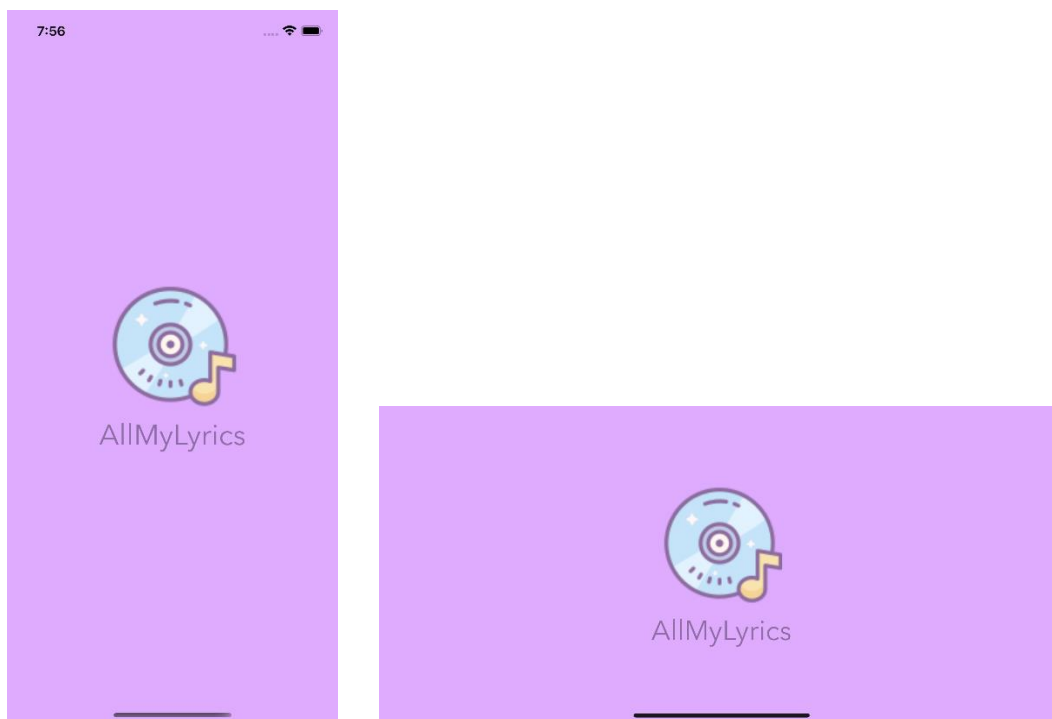
Además de todas estas herramientas, también ha sido necesario consultar diferentes fuentes como **StackOverflow**, otros foros de las herramientas usadas y repositorios de **GitHub** para resolver dudas sobre la programación y diseño de la aplicación.

4. Descripción funcional de la aplicación

El objetivo principal de la aplicación, como se ha comentado en apartados anteriores es proporcionar al usuario un motor de búsqueda de letras de canciones, la posibilidad de añadir las letras manualmente y almacenarlas en la base de datos local. La aplicación cuenta con 4 vistas o pantallas en total que se describirán a continuación:

- PANTALLA DE ARRANQUE

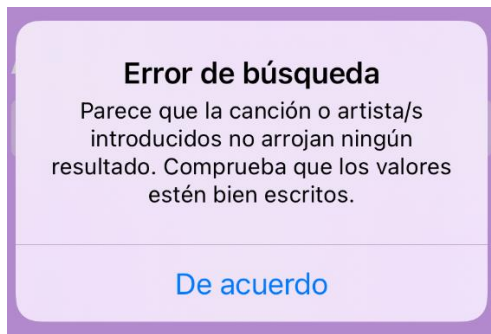
En la pantalla de arranque simplemente se ha incluido el icono de la aplicación junto con el título de la aplicación.



Vista vertical y horizontal de la pantalla de arranque

- PANTALLA PRINCIPAL / DE BÚSQUEDA

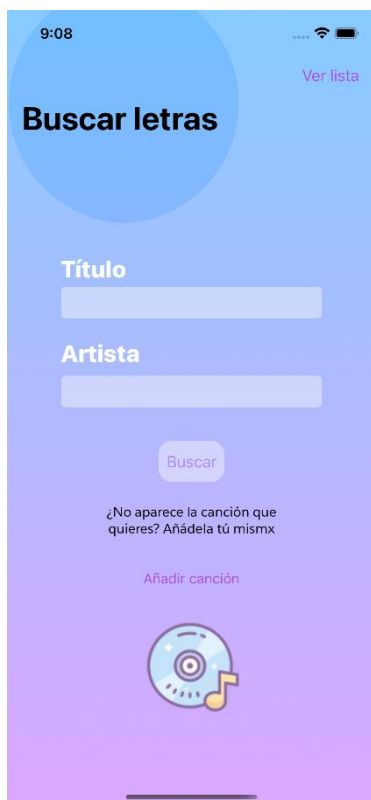
La primera pantalla que se muestra es la de búsqueda de canciones. En ella el usuario puede buscar escribiendo en los campos la información necesaria para realizar la petición a la API: el título de la canción y el nombre del artista. Pulsando el botón “Buscar” y si se encuentra un resultado para los valores



introducidos, se redirigirá a la página donde se muestran las letras (ver página de letras). Si los valores no devuelven ningún resultado (no está en la API), aparecerá una notificación avisando al usuario de que no se encuentra esa canción y que revise los datos en caso de que estén mal escritos.

Además de esta opción de búsqueda en esta pantalla se puede acceder a la pantalla de añadir canción pulsando el botón "Añadir canción", donde el usuario podrá introducir manualmente los datos de la canción y almacenarla en la base de datos de la aplicación.

En la parte superior de la aplicación aparece un navegador en todas las páginas donde se permite al usuario volver a la anterior pantalla o acceder a otras opciones. En el caso de esta pantalla se puede acceder a la lista donde se muestran todas las canciones (ver pantalla de lista).



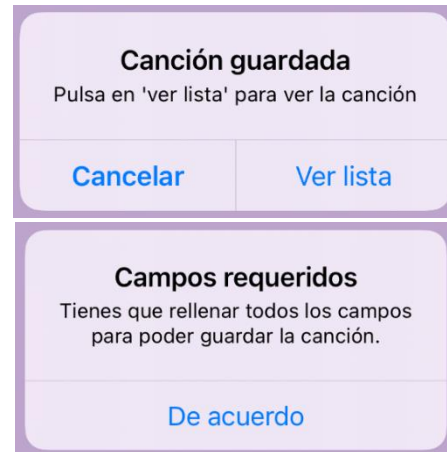
Vistas de la página de Buscar canción



- PANTALLA DE AÑADIR CANCIÓN

En esta pantalla el usuario podrá introducir manualmente los datos de una canción y guardarla en la base de datos local en caso de que el buscador de canciones no retornara resultados de dicha canción.

Para ello se han de rellenar obligatoriamente los campos de “título”, “artista/s” y el de la letra de la canción. Al pulsar el botón de guardado aparecerá una notificación que avisa de que esa canción se ha almacenado y dará la opción de ver dicha canción en la lista de canciones. En caso de que falte algún campo por cumplimentar aparecerá una alerta avisando de ello y la canción no se guardará.



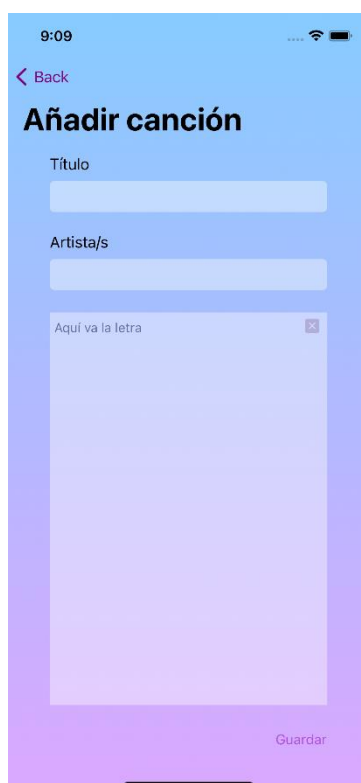
Canción guardada
Pulsa en 'ver lista' para ver la canción

Cancelar Ver lista

Campos requeridos
Tienes que rellenar todos los campos para poder guardar la canción.

De acuerdo

A esta página de la aplicación se puede acceder desde la pantalla de búsqueda y la pantalla de la lista de canciones. Esta página a su vez permite redirigir al usuario a la lista de canciones.



9:09

< Back

Añadir canción

Título

Artista/s

Aquí va la letra

Guardar

Vistas de la página de Añadir canción



< Back Añadir canción

Título

Artista/s

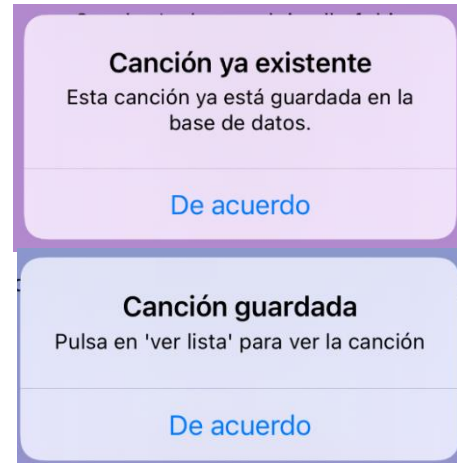
Aquí va la letra

Guardar

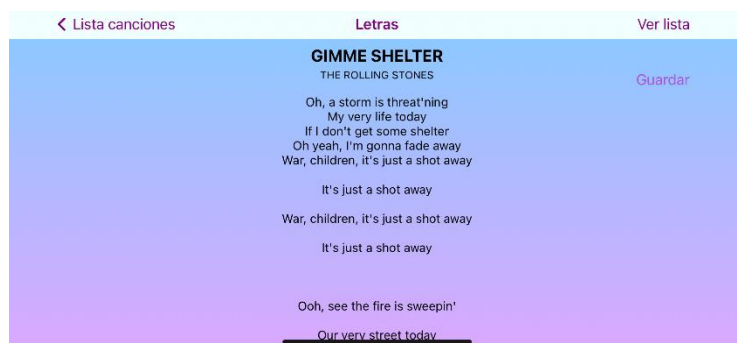
- PANTALLA DE VISUALIZAR LETRA

En esta pantalla se puede ver la letra de la canción que el usuario ha buscado en la pantalla de búsqueda o ha seleccionado en la lista de canciones. Esta vista además permite guardar la canción en la base de datos local, en caso de que esta no esté almacenada ya en la lista de canciones.

Una vez que se pulse el botón de guardado aparecerá una notificación avisando de que se ha guardado la canción con éxito y de que ya se puede visualizar en la lista. Si esta canción ya estuviera guardada nos aparecería también una notificación avisando de que no se puede almacenar duplicada. En caso de que se acceda a esta vista desde la pantalla de la lista, el botón de guardar aparecerá desactivado.

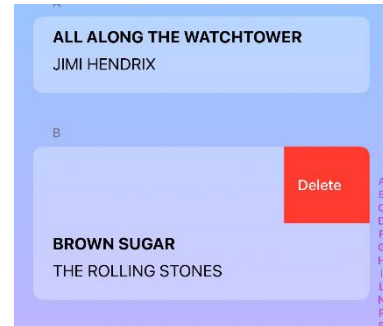


Vistas de la página de Visualizar letra



- PANTALLA DE LISTA DE CANCIONES

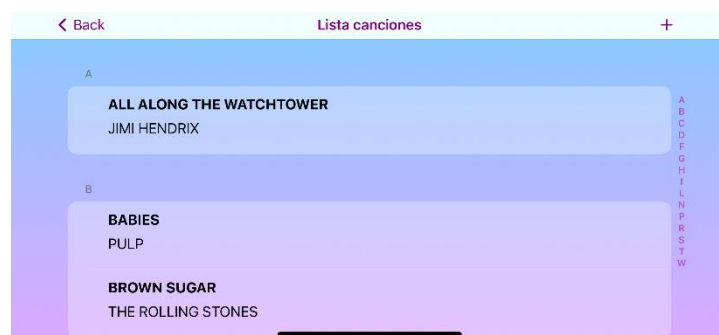
En esta pantalla se muestra la lista de canciones que el usuario ha guardado en la base de datos de la aplicación. Estas aparecen ordenadas alfabéticamente según el título y separadas por secciones por letra. Pulsando en cualquiera de estas canciones se podrá acceder a la vista de la letra de la canción pulsada. Además de esto, el usuario podrá eliminar una canción deslizando hacia la izquierda en la celda de la canción que se quiera eliminar. Al realizar este gesto aparecerá la celda de color rojo y el usuario tendrá que pulsar el botón de “Delete”.



Pulsando el botón “+” que aparece en el navegador de la parte superior se puede acceder a la pantalla de añadir canción.



Vistas de la página de la Lista de canciones



5. Descripción técnica de la aplicación

- Backend, llamada a la API

La parte del Backend en esta aplicación es muy sencilla, ya que no se requiere hacer más que una llamada a la API usada para obtener los datos deseados (<https://lyricsovh.docs.apiary.io/#>). Consta de una clase, `APILyrics`, que se encarga de hacer una petición HTTP a la API con los parámetros necesarios para realizar la búsqueda (en este caso el título y el artista de la canción). Este API devolverá un objeto JSON con la letra de la canción solicitada si esta se encuentra, y si no, devolverá error. Para parsear el objeto JSON recibido de la petición se ha usado `Decodable`, ya que es muy fácil de usar.

La búsqueda en la API tiene el siguiente formato:

`GET https://api.lyrics.ovh/v1/artist/title`

```
import Foundation
import Alamofire

struct Resultado: Decodable {
    var lyrics: String
}

class APILyrics {
    var session = Session.default

    func pedirLyrics(artista: String, cancion: String, completion: @escaping (Result<Resultado, AFError>->()) {
        session.request("https://api.lyrics.ovh/v1/"+artista+"/"+cancion, method: .get).validate().responseDecodable(of: Resultado.self) { (resultado) in completion(resultado.result)
    }
}
```

Clase APILyrics.swift

El resto de la aplicación trata de seguir el patrón **MVC**, ya intrínseco en la estructura de las aplicaciones de iOS.

- **Modelo: Core Data.**

El modelo de datos persistente de la app está compuesto por el framework de persistencia Core Data que contiene las siguientes clases y entidades

- **Song:** contiene la información de la canción, sus atributos son:
 - Title: (String) el título de la canción.
 - Lyrics: (String) la letra de la canción.

Song está relacionada con Artist mediante la relación **SongArtist**, que es de tipo Many to One

- **Artist:** contiene la información del artista, su único atributo es:
 - Name (String): el nombre del artista

Artist está relacionado con Song mediante la relación **ArtistSong**, que es de tipo One to Many.

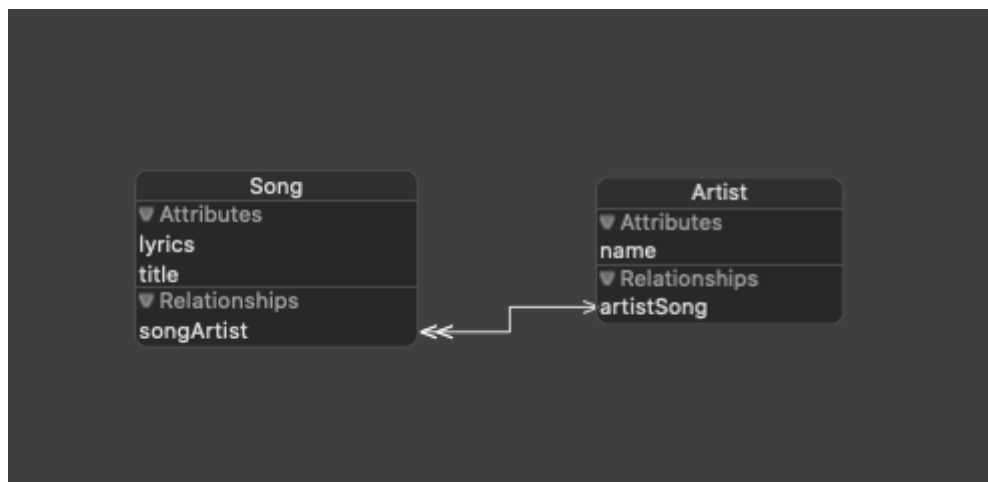


Tabla relacional del modelo de datos [Model.xcdatamodeld]

De estos “objetos” `NSManagedObject` se han obtenido automáticamente sus subclases en código, para poder instanciarlos y realizar acciones con ellos desde el controlador ya que me parece la manera más sencilla y ordenada para acceder a ellos. Cada objeto tiene dos ficheros, uno define a la clase y el otro las propiedades, estos siguen la estructura de:

`Nombreobjeto+CoreDataClass.swift`

`Nombreobjeto+CoreDataProperties.swift`

Además de estas clases se ha creado una aparte (`CoreDataStack.swift`) para manejar más fácilmente el modelo de datos que hemos creado. En esta se definen e instancian otras clases necesarias en el modelo de datos, como `NSManagedObjectModel`, `NSManagedObjectContext` y `NSPersistentStoreCoordinator`, para poder hacer uso de ellos en los controladores.

- **Vista: UIKit** (Storyboards)

En mi caso he decidido usar UIKit con Storyboards porque es mi primer contacto con el mundo de iOS y las aplicaciones móviles y me parecía una manera mucho más intuitiva de programar teniendo en cuenta mis conocimientos previos.

Las clases Storyboard son dos y están compuestas por: la pantalla de arranque y el storyboard como tal de la aplicación ya activa.

A la pantalla de arranque solo le añadí otro color al fondo y el logo de la aplicación junto con su nombre.

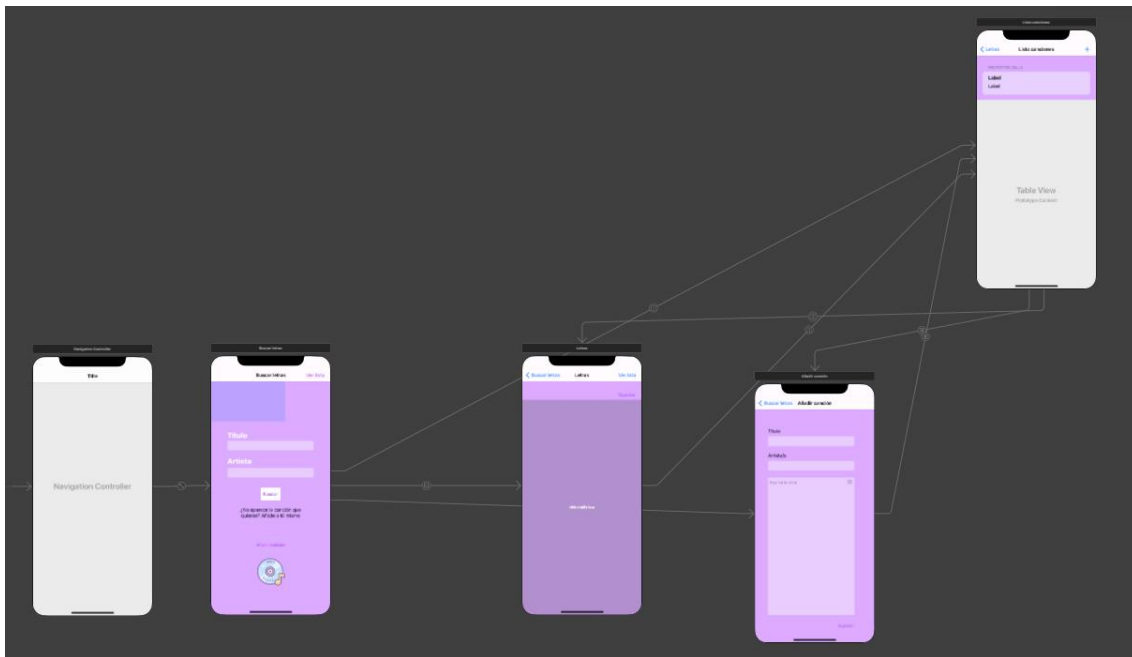
Esta pantalla como el resto están hechas para que sean compatibles con cualquier orientación en la que el dispositivo se encuentre: horizontal o vertical.

El Storyboard de la aplicación está compuesto por 4 pantallas como se ha mencionado en otro apartado:

- Búsqueda de canciones
- Añadir canciones manualmente
- Visualización de letras
- Listado con las canciones

Todas ellas se encuentran controladas por un Navigation Controller, el cual nos permite el poder navegar entre pantallas y que se cree un flujo de aplicación consistente. Además de esto, se han añadido otro tipo de accesos o “atajos” a otras pantallas a través de botones con segues incluidos en la barra de navegación o en otras partes de la aplicación.

La mayoría de los componentes de diseño que tiene esta aplicación han sido añadidos desde el Storyboard y no programados, incluidos los márgenes y constantes de los elementos y la app en sí.



Storyboard con el flujo de la aplicación entre pantallas

- **Controlador: clases Swift** controladores para cada pantalla.

La parte de los controladores está compuesta por cuatro ViewControllers programados en Swift, uno por cada vista de la aplicación.

- **ViewController.swift:**

Consiste en el controlador inicial y de la vista de búsqueda de canciones. Este contiene dos UITextField que recogen el valor del título y el artista de la canción que el usuario introduzca (estos son codificados a un formato apto para URLs) y pulsando el botón se realiza la llamada a la clase de la API con los parámetros de las canciones ya codificados, que devuelve el JSON con las letras y realiza un segue pasando los parámetros de la canción a la siguiente página que es la de mostrar letra. Además de esta acción, también se puede acceder directamente a la página de añadir canción desde el botón “Añadir canción” o a la página de la lista desde el botón de la barra de navegación “Ver lista”. En todos estos segues también es necesario pasar la instancia de CoreDataStack para tener los datos persistentes actualizados a lo largo de la aplicación.

- **LyricsViewController.swift:**

A este controlador se accede tanto desde el buscador como desde la página de la lista de canciones. Los segues de las vistas que derivan en esta pasan la información de la canción (título, artista y letra) y esta se muestra usando UILabels para cada parámetro. Además, la letra se muestra usando un UIScrollView ya que es necesario realizar scroll visualizarla entera.

Esta vista además permite almacenar la canción si esta no se encuentra en la base de datos. Para saber si se encuentra ya se hace una consulta con NSFetchedRequest y NSPredicate, donde se cuenta el número de apariciones de una canción en la base de datos que tenga la misma letra. Si este numero es 0 la canción se guarda, si no, aparecerá una alerta.

El proceso de guardado consiste en crear entidades de cada clase (Song y Artist) y guardar la información sobre sus atributos que estaban almacenados en las respectivas UILabels.

```
@IBAction func save(_ sender: Any) {

    let entitySong = NSEntityDescription.entity(forEntityName: "Song",
                                                in:managedContext)!
    let entityArtist = NSEntityDescription.entity(forEntityName: "Artist",
                                                  in:managedContext)!

    let fetchRequest = NSFetchedRequest<NSFetchRequestResult>(entityName: "Song")
    let predicate = NSPredicate(format: "lyrics == %@", lyrics)
    fetchRequest.predicate = predicate
    fetchRequest.fetchLimit = 1

    do {
        let count = try managedContext.count(for: fetchRequest)
        if count == 0 {
            //guardar objeto en bbdd
            let song = Song(entity: entitySong,
                            insertInto: managedContext)
            let artist = Artist(entity: entityArtist,
                                insertInto: managedContext)

            artist.name = self.artistaLabel.text
            song.title = self.tituloLabel.text
            song.lyrics = self.display.text
            song.songArtist = artist

            CoreDataStack.saveContext()

            showAlert(title: "Canción guardada", message: "Pulsa en 'ver lista' para ver la canción")
        }else{ //si ya existe

            showAlert(title: "Canción ya existente", message: "Esta canción ya está guardada en la base de datos.")
        }
    } catch let error as NSError {
        print("Could not save \(error), \(error.userInfo)")
        showAlert(title: "Error", message: "No se ha podido guardar la canción en la base de datos.")
    }
}
```

Fragmento del código de guardado de una canción

- **AddViewController.swift:**

En esta vista los valores del título y el artista que el usuario introduce son recogidos por UITextField. El campo de la letra es un UITextView ya que este nos ofrece introducir texto de más de una línea. Una vez que el usuario da la señal de guardar pulsando el botón se realizan las comprobaciones pertinentes, en este caso comprobar que todos los campos han sido completados y que la canción no esté almacenada ya en la base de datos, como se explicó en la vista anterior. Si se da alguno de estos casos saltará una notificación. En caso contrario, la canción se guarda y aparecería una alerta avisando de que la canción ha sido guardada dando la opción de redirigir al usuario para ver dicha canción en la lista de canciones

- **ListViewController.swift:**

La vista con la lista de canciones es la más compleja de todas ya que requiere cargar todas las canciones de la base de datos en una lista UITableView y que además esté actualizada. En este caso, para mostrar las canciones se hace una NSFetchRequest del elemento Song y se llama al NSSortDescriptor para ordenar los valores alfabéticamente. Además de esto y dado que cada canción aparece en una celda y estas en secciones ordenadas según el título es necesario realizar manualmente esta organización por secciones ya que cada sección variará según el número de canciones que empiecen por x letra. En este caso, para estructurar mejor los datos se creó un diccionario del tipo *[String: [Song]]* donde cada letra del alfabeto tendría asignada un array de canciones que empezarían por dicha letra. Ejemplo:

- ["a": [cancion1, cancion2], "b": [cancion3], "c": [cancion4]...]

```

@IBOutlet weak var tableView: UITableView!
let gradientLayer = CAGradientLayer()
var canciones: [Song] = []
var songsDictionary = [String: [Song]]() // ["a": [asong, asong2],...]
var songSectionTitles = [String]() // ["a", "b", ...]
var CoreDataStack: CoreDataStack!
var managedContext: NSManagedObjectContext!
{
    get
    {
        return CoreDataStack.context
    }
}

```

```

let fetchRequest = NSFetchRequest<Song>(entityName: "Song")
let songSortDescriptor = NSSortDescriptor(key: "title", ascending: true, selector: #selector(NSString.localizedCompare(_)))
do {
    fetchRequest.sortDescriptors = [songSortDescriptor]
    canciones = try managedContext.fetch(fetchRequest)

    for cancion in canciones {
        let primeraLetra = String((cancion.title?.prefix(1))!)
        if var songValues = songsDictionary[primeraLetra] {
            songValues.append(cancion)
            songsDictionary[primeraLetra] = songValues
        } else {
            songsDictionary[primeraLetra] = [cancion]
        }
    }

    songSectionTitles = [String](songsDictionary.keys)
    songSectionTitles = songSectionTitles.sorted(by: {$0 < $1})
} catch let error as NSError {
    print("Could not fetch \(error), \(error.userInfo)")
}

```

Fragmento del código donde se crea el diccionario con las canciones

De esta manera será más fácil retornar los valores adecuados para cada método de UITableView como *numberOfSections*, *titleForHeaderInSection*, *sectionIndexTitles*, *numberOfRowsInSection*, *cellForRowAt...* ya que es necesario que cuando se pulse en cada celda se redirija al usuario a la pantalla de vista de la letra de la canción correspondiente y enviar por segue los atributos de la canción. Además de eso, también se ha añadido la función de hacer swipe hacia la izquierda en las celdas para eliminar canciones de la base de datos. Las celdas están customizadas y tienen su propia clase ya que cuentan con dos valores que las identifican y que han de mostrarse: el título y el artista de la canción.

- AppDelegate y SceneDelegate

Estos delegados generalmente se utilizan para manejar el ciclo de vida de la aplicación o como sitio donde declarar variables, CoreDataStack, por ejemplo. En mi caso, el SceneDelegate ha sido eliminado y sustituido por el AppDelegate ya que no se requería rellenar ninguno de los métodos vacíos que contiene. Además, he editado la barra de navegación.

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, UIWindowSceneDelegate {

    lazy var coreDataStack = CoreDataStack()
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        coreDataStack.saveContext()
    }

    func applicationWillTerminate(_ application: UIApplication) {
        coreDataStack.saveContext()
    }

    // MARK: UISceneSession Lifecycle
    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {
        let navigationController = window!.rootViewController as! UINavigationController
        let viewController = navigationController.topViewController as! ViewController
        viewController.coreDataStack = coreDataStack
        navigationController.navigationBar.barStyle = .default
        navigationController.navigationBar.isTranslucent = true
        navigationController.navigationBar.titleTextAttributes = [.foregroundColor: UIColor.purple]
        navigationController.navigationBar.tintColor = .purple //controles del navigator
        navigationController.navigationBar.prefersLargeTitles = true
    }

    func application(_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {
        // Called when a new scene session is being created.
        // Use this method to select a configuration to create the new scene with.
        return UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)
    }

    func application(_ application: UIApplication, didDiscardSceneSessions sceneSessions: Set<UISceneSession>) {
        // Called when the user discards a scene session.
        // If any sessions were discarded while the application was not running, this will be called shortly after application:didFinishLaunchingWithOptions.
        // Use this method to release any resources that were specific to the discarded scenes, as they will not return.
    }
}
```

Código de AppDelegate.swift

- Información general de la aplicación

La app ha sido programada y probada en el simulador de iPhone 12 teniendo en cuenta las dimensiones de este dispositivo y otro tipo de características como los márgenes irregulares (safe space) que poseen ahora los nuevos modelos de iPhone.

La versión del sistema operativo para la que se ha hecho la aplicación es la versión 14.4, aun así, esta podría ser desplegada en dispositivos de mínimo iOS 13.0, ya que las versiones anteriores no soportan UIScene y otros elementos.

6. Análisis crítico de la aplicación

Este proyecto ha sido mi primer contacto con el software de Apple y estoy muy satisfecha con todo lo que he aprendido con él. El proceso de aprendizaje con la aplicación ha sido muy interesante ya que es muy interactivo e intuitivo.

Al principio no sabía bien qué sería capaz de llegar a implementar para esta aplicación, por eso decidí empezar por una base y de ahí intentar hacer la aplicación más grande. Al final vi que era más fácil de lo que pensaba. Aun así, me he tenido que enfrentar a problemas que me han llevado bastante tiempo resolver, como por ejemplo ajustar las constraints para el diseño de la aplicación y sus componentes o el conseguir poner en marcha la “base de datos” de la aplicación. Gracias a las tutorías y a los foros y repositorios de código en internet he sido capaz de resolver todas las dudas que me han ido surgiendo a lo largo del proyecto.

La principal debilidad que este proyecto tiene es el diseño y la estabilidad de la aplicación, puesto que yo misma considero que no es una aplicación que sea lo suficientemente segura como para “comercializarla” o poder publicarla en la App Store. El código probablemente necesitase una refactorización y mejora, además de añadir comprobaciones que hicieran la app más estable.

Otra cosa a mejorar sería la compatibilidad de la aplicación con las diferentes dimensiones de pantallas para diferentes modelos de iPhone o de iPad, puesto que esta aplicación no es totalmente “responsive” y solamente está adaptada para el tamaño de un iPhone 12.

Mi conclusión final sobre la aplicación es que he conseguido realizar exactamente lo que pretendía obtener teniendo en cuenta que es mi primer proyecto con iOS y estoy bastante contenta con el resultado final, pese a ser simple y tener algunos defectos que se podrían mejorar en el futuro.

7. Bibliografía

API

- Lyrics.ovh: <https://lyricsovh.docs.apiary.io/#>

COCOAPODS

- CocoaPods: <https://cocoapods.org/>
- Alamofire: <https://cocoapods.org/pods/Alamofire>
- DesignSystem: <https://cocoapods.org/pods/DesignSystem>

RECURSOS

- *Indexed Table View iOS Tutorial (ioscreator):*
<https://www.ioscreator.com/tutorials/indexed-table-view-ios-tutorial-ios11>
- Icono aplicación: <https://icons8.com/icon/46688/music-record>
- *Swift Tutorials: How to create iPhone apps that support both orientations portrait and landscape (YouTube):*
<https://www.youtube.com/watch?v=NP5LL-MPJO8>
- *Swift Gradient in 4 lines of code (betterprogramming):*
<https://betterprogramming.pub/swift-gradient-in-4-lines-of-code-6f81809da741>