



INSTITUTO TÉCNICO
DE ESTUDIOS
PROFESIONALES

SwiftUI

iOS Development





Tutorial de desarrollo de iOS con Swift

Tutorial de desarrollo de iOS El uso de Swift proporciona los conceptos básicos y avanzados del desarrollo de iOS. Nuestro tutorial de desarrollo de iOS está diseñado tanto para principiantes como para profesionales.

Prerrequisitos

Antes de aprender el tutorial de desarrollo de iOS, debe tener los conocimientos básicos sobre la forma en que funciona un sistema informático y el lenguaje de programación.

Audiencia

Nuestro tutorial de desarrollo de iOS está diseñado para ayudar a principiantes y profesionales.

Introducción a XCode IDE

XCode es un entorno de desarrollo integrado desarrollado para funcionar en sistemas operativos Mac. Contiene un conjunto de herramientas de desarrollo de software desarrolladas por apple. XCode nos facilita el desarrollo de software para macOS, tvOS, iOS y watchOS. La última versión estable de XCode es 11.0, que está disponible en la Mac App Store para todos los usuarios de macOS Mojave. En esta sección del tutorial, veremos varios contextos de XCode. También veremos varias secciones de XCode.

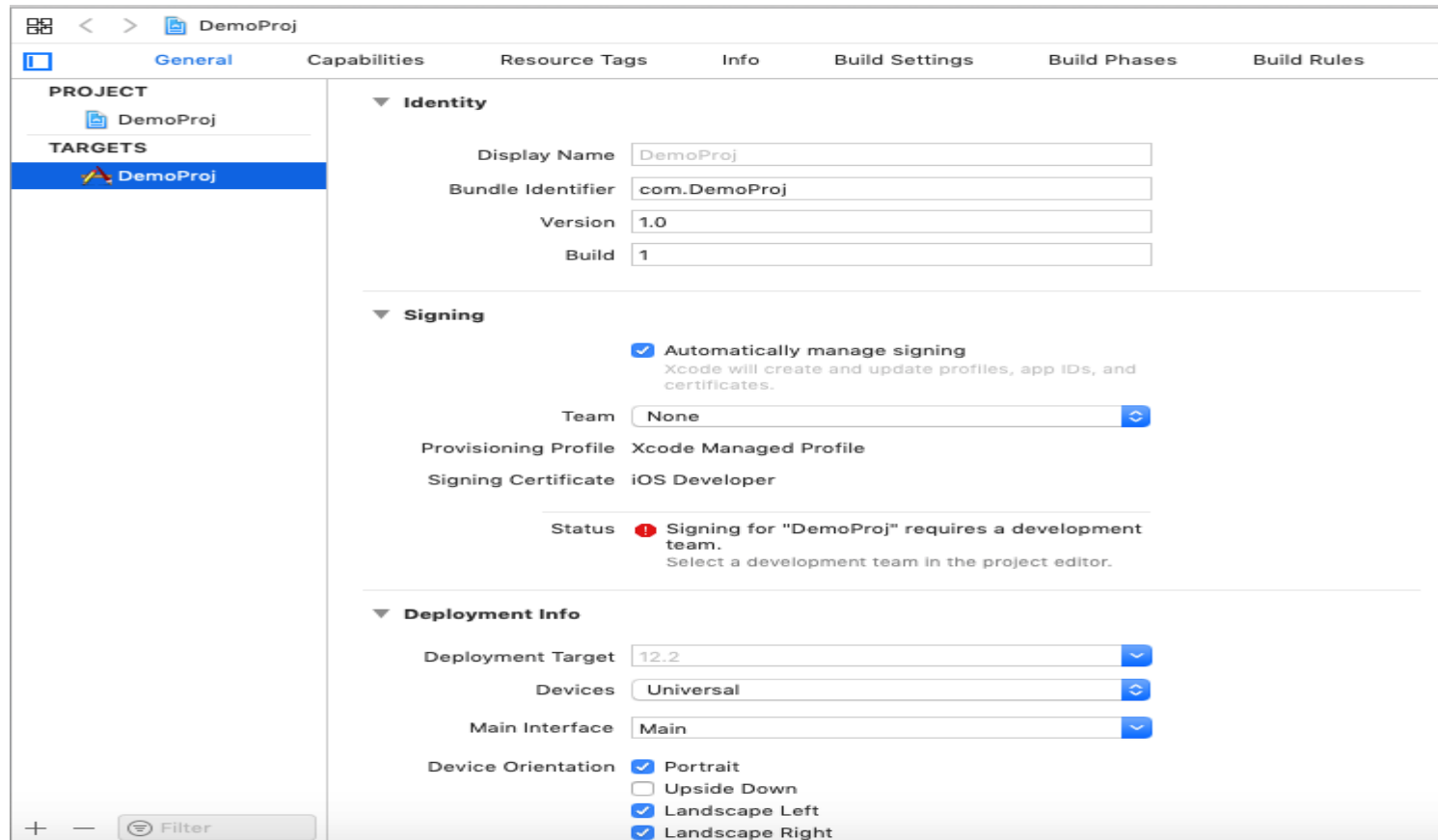
Historial de versiones

SN	Año de lanzamiento	Sistema operativo	Características
Serie 1.x	2003	MacOS 10.3 +	Se basa en el creador del proyecto. XCode 1.5 tiene un depurador mejorado y un mejor compilador de código.
Serie 2.X	2005	MacOS 10.4 +	Incluía Quartz Composer, una mejor indexación de sentido de código para Java y compatibilidad con Ant. XCode 2.1 podría crear archivos binarios precompilados.
3.X series	2007	macOS 10.5+	La serie XCode 3.X incluye la herramienta de depuración DTrace (instrumentos), soporte de refactorización, documentación sensible al contexto y Objective C 2.0 con recolección de basura.
Serie 4.X	2011	macOS 10.6.8+	XCode versión 4 integró la herramienta de edición XCode y el generador de interfaces en una sola aplicación. Entre muchos cambios, también incluyó el soporte para iOS iOS 5.1, mejoras al simulador de iOS, y sugirió el cambio al depurador LLDB frente al depurador GDB.
Serie 5.X	2013	macOS 10.8+	Agregó soporte para iOS 7 SDK. También agregó una versión de Clang que genera código ARM de 64 bits para iOS 7. Apple eliminó el soporte para construir binarios de Cocoa recolectados en XCode 5.1
6.x series	2014	macOS 10.9.4+	La versión 6 de XCode proporcionó muchas mejoras, incluida la compatibilidad con todos los nuevos lenguajes de programación de apple, es decir, Swift. XCode 6 también incluye soporte para parques infantiles y herramientas de depuración en vivo.
Serie 7.x	2015	macOS 10.10.3+	XCode versión 7 proporcionó soporte para Swift 2 y metal para OS X. También agregó soporte para implementar un dispositivo iOS sin tener una licencia de desarrollador de Apple.
Serie 8.x	2016	macOS 10.11.5+	XCode versión 8 proporcionó soporte para Swift 3.
Serie 9.x	2017	macOS 10.12.6+	Proporcionó soporte para Swift 4 y metal 2 para OS X.
10.x series	2018	macOS 10.13.6+	Xcode 10 introdujo el soporte para el modo oscuro anunciado para macOS Mojave, las plataformas de colaboración Bitbucket y GitLab (además de GitHub), los modelos de entrenamiento de aprendizaje automático de parques infantiles y las nuevas características en Swift 4.2 y Metal 2.1, así como mejoras en el editor y el sistema de construcción del proyecto.
11.X series	2019	macOS 10.14.4+	XCode 11 introdujo soporte para las nuevas características en Swift 5.1, así como el nuevo marco SwiftUI (aunque las herramientas interactivas de la interfaz de usuario solo están disponibles cuando se ejecutan bajo macOS 10.15). También admite la creación de aplicaciones de iPad que se ejecutan en macOS; incluye soporte integrado para Swift Package Manager; y contiene mejoras adicionales para el editor, incluido un "minimapa" que ofrece una visión general de un archivo de código fuente con navegación rápida.

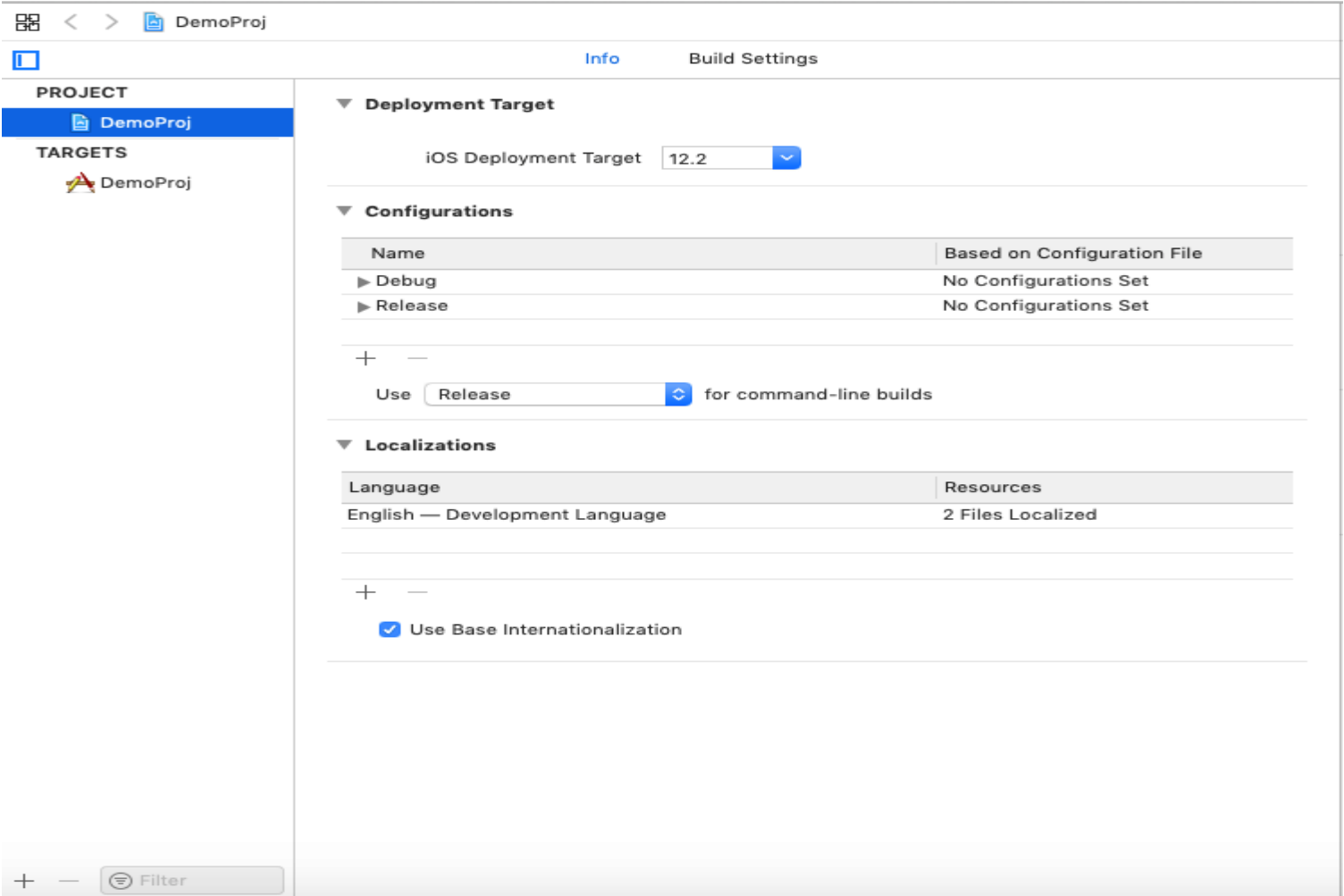
Un paseo rápido de XCode

Echemos un vistazo rápido a XCode 10.2.1 y comprendamos cómo se realizará el desarrollo en XCode.

Cuando creamos un nuevo proyecto XCode, se muestra la siguiente ventana que proporciona la información de destino del proyecto XCode. Muestra toda la información del proyecto que incluye el Identificador de paquete, la versión de la aplicación, la versión de compilación, la información de firma, la información de implementación, los binarios vinculados y la información del marco, y los iconos de inicio de la aplicación.

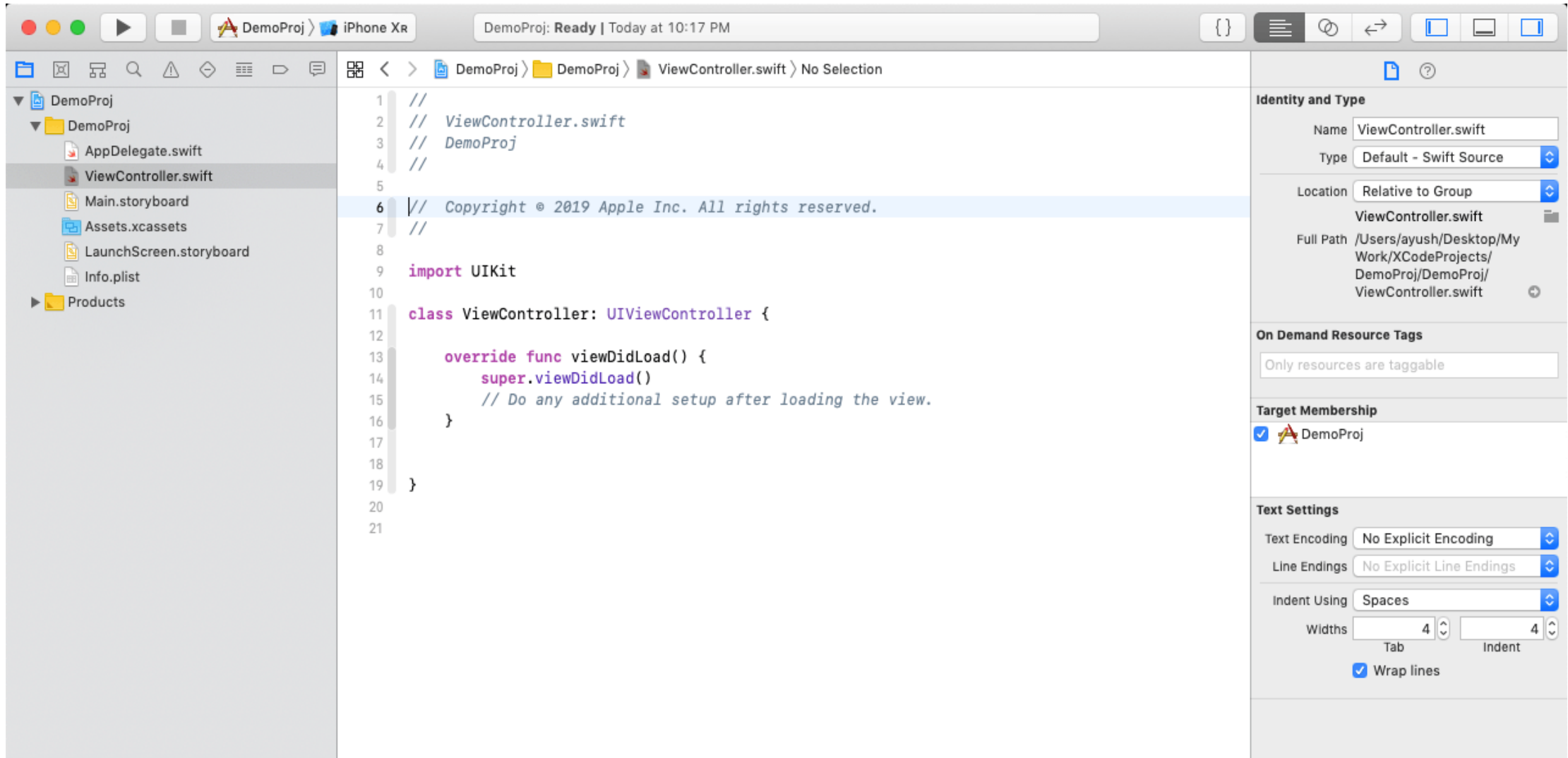


Encima de la información de destino, hay un panel de información del proyecto que muestra toda la información sobre el proyecto dada en la siguiente imagen. Contiene información sobre la versión de iOS para la que se creó la aplicación. También contiene información de lanzamiento



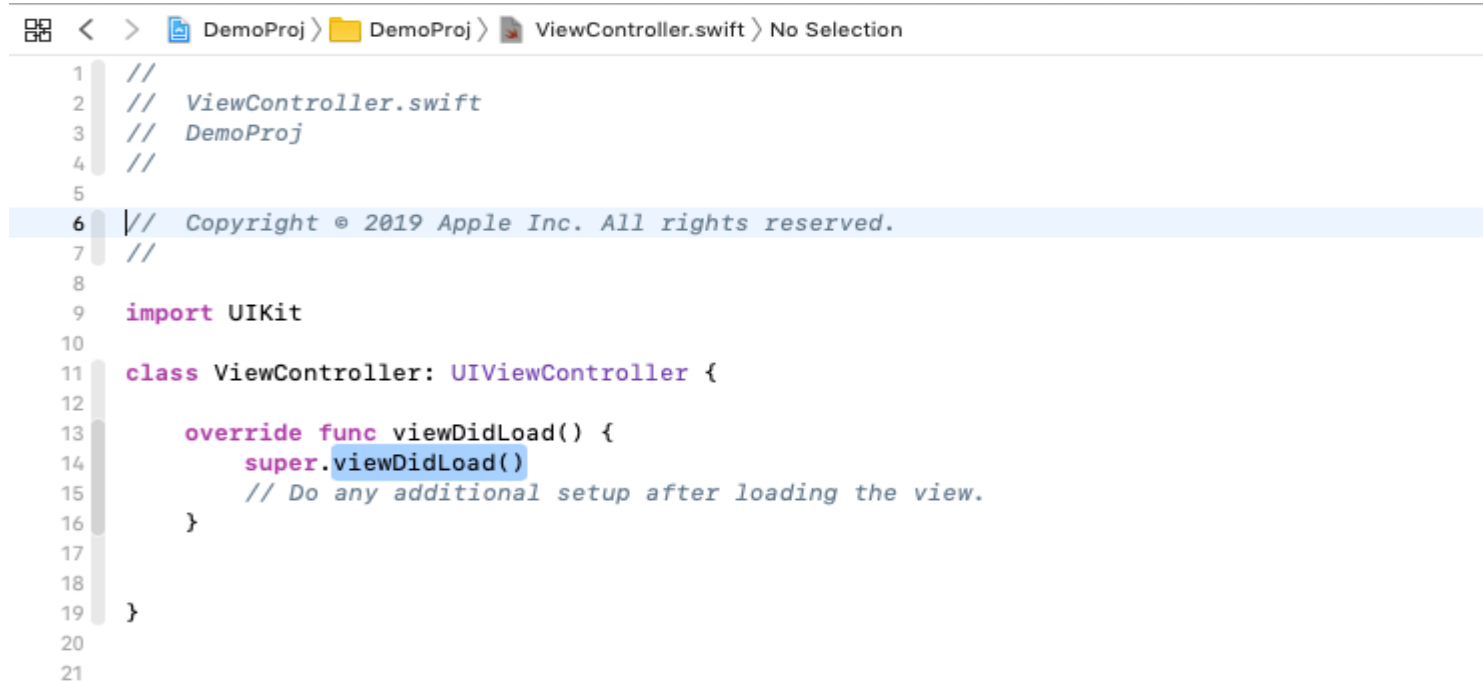
Discutiremos el objetivo y la información del proyecto más adelante en este tutorial en detalle.

Ahora echemos un vistazo a todos los componentes de una ventana XCode (debajo de la imagen) en detalle.



Editor estándar

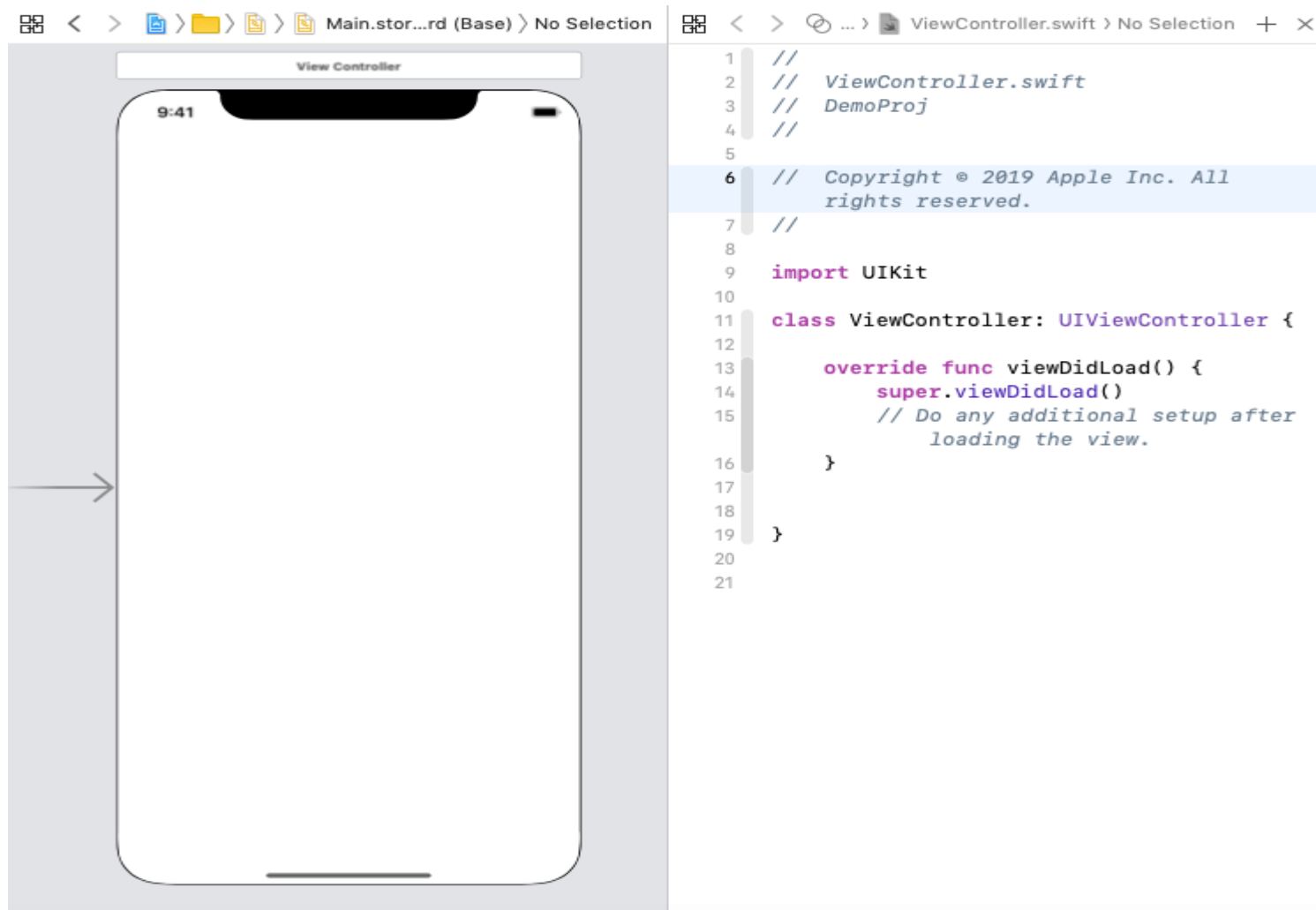
El editor estándar se muestra en el medio de la ventana. Como su nombre indica, es el editor estándar del proyecto en el que se editan los archivos del proyecto. Contiene la información sobre el archivo comentado en la parte superior y el archivo de clase de View Controller inicial con el método de ciclo de vida creado. (Los métodos del ciclo de vida se analizarán más adelante en este tutorial. En el panel superior del editor Estándar, se muestra la información jerárquica sobre el archivo del proyecto. También se puede usar para abrir otros archivos en el mismo editor. Sin embargo, también podemos navegar a otros archivos de proyecto usando el navegador de proyectos también.



```
1 //  
2 // ViewController.swift  
3 // DemoProj  
4 //  
5  
6 // Copyright © 2019 Apple Inc. All rights reserved.  
7 //  
8  
9 import UIKit  
10  
11 class ViewController: UIViewController {  
12  
13     override func viewDidLoad() {  
14         super.viewDidLoad()  
15         // Do any additional setup after loading the view.  
16     }  
17  
18  
19 }  
20  
21
```

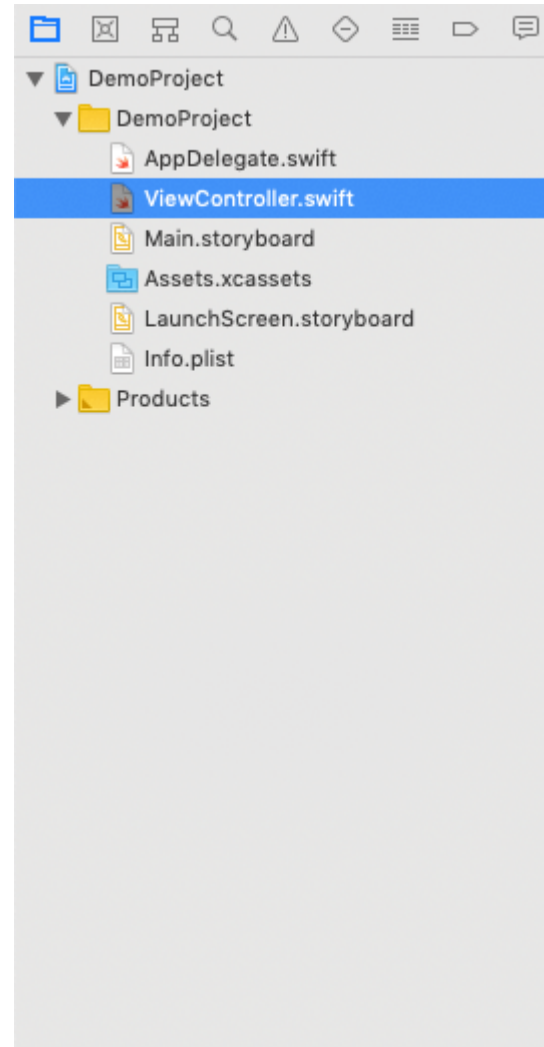
Editor asistente

El Editor Asistente se utiliza principalmente para crear salidas de los componentes del guión gráfico (campo de texto, etiqueta, etc.) en el archivo de clase del controlador de vista correspondiente. Sin embargo, el editor Asistente nos facilita mirar dos archivos en el editor simultáneamente.



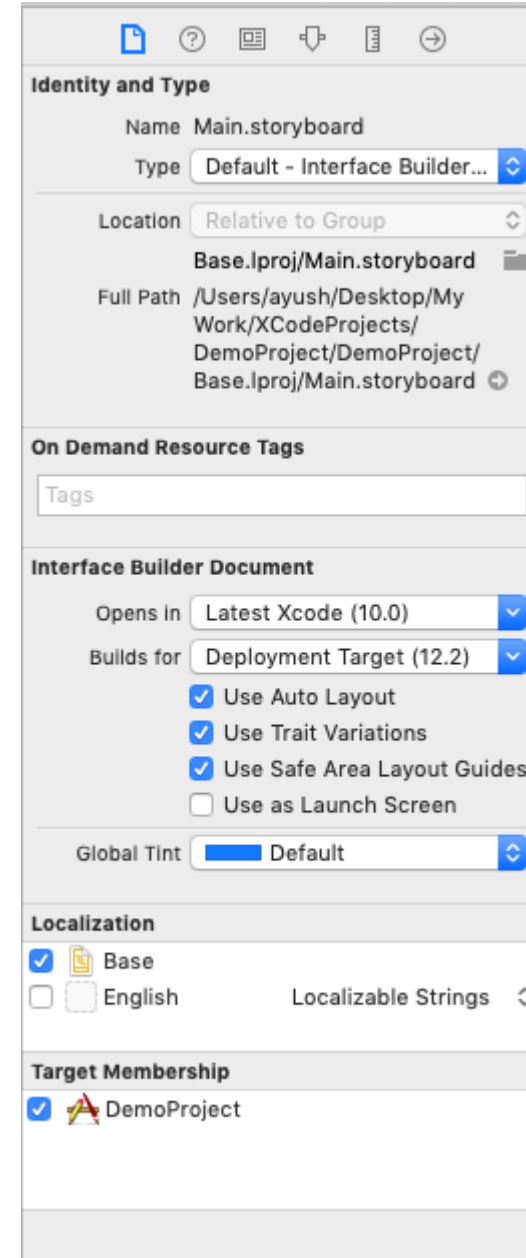
Navegador de proyectos

El navegador del proyecto se muestra a la izquierda de la ventana. Muestra la estructura de archivos del proyecto. Se utiliza para navegar por el proyecto. Inicialmente, un proyecto XCode contiene los archivos que se muestran en la siguiente imagen.



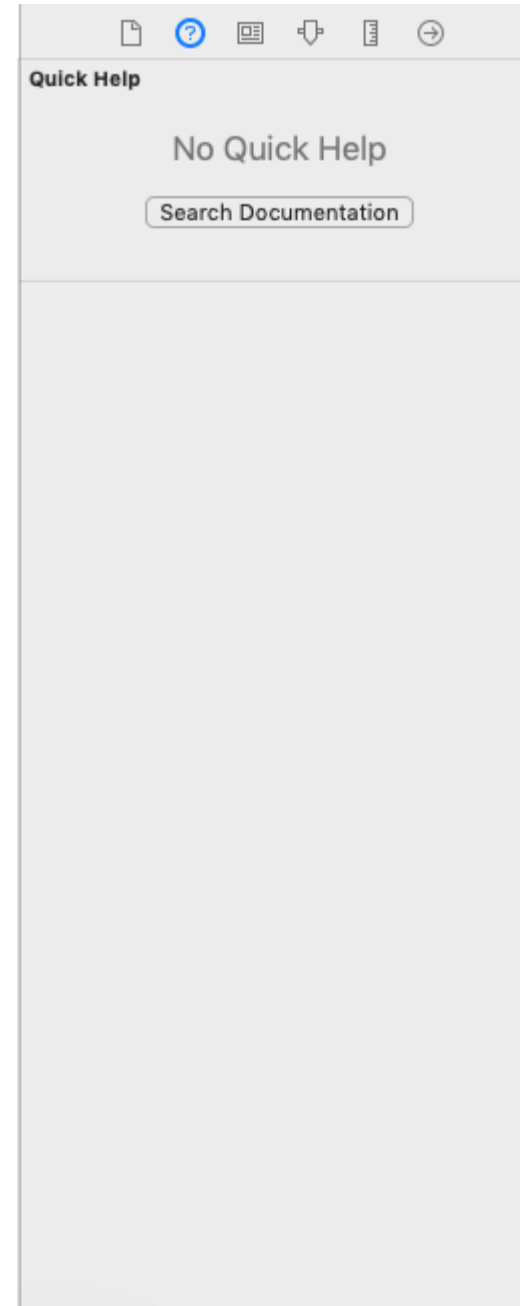
Inspector de archivos

Los inspectores se muestran a la derecha de la ventana XCode, como se muestra en la imagen a continuación. El inspector de archivos muestra la información completa sobre el archivo rápido correspondiente abierto en el editor estándar. Contiene el nombre, el tipo, la ubicación y la información relacionada con el documento del creador de interfaces.



Inspector de ayuda rápida

Se utiliza para proporcionar ayuda al usuario donde puede buscar la documentación sobre la sintaxis. Se muestra como la imagen de abajo.



Inspector de identidad

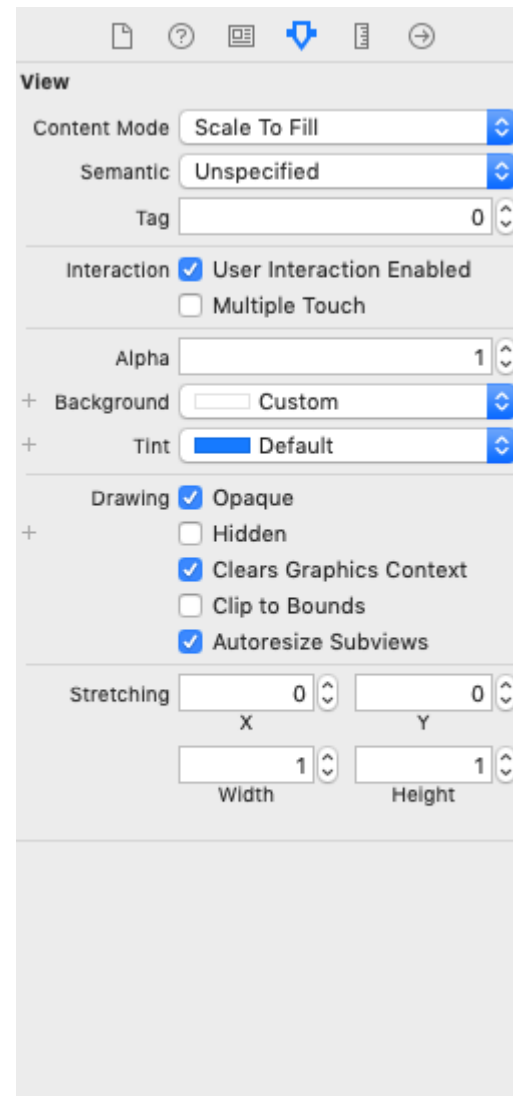
Identity Inspector se usa principalmente cuando necesitamos trabajar con el guión gráfico. Muestra información sobre los componentes del guión gráfico (View Controllers) y sus correspondientes archivos de clase Swift. Para programar los componentes del guión gráfico en consecuencia, debemos asignarles archivos de clase. El inspector de identidad muestra toda esta información, como se muestra en la imagen a continuación. Solicita al desarrollador que asigne la clase y el módulo al controlador de vista del guión gráfico correspondiente. También le pide al desarrollador que asigne el nombre de identidad al controlador de vista, que se utiliza en todo el proyecto para identificar el controlador de vista. También podemos dar algunas restricciones de tiempo de ejecución a la vista de UIV que se muestra en el guión gráfico.

The screenshot displays the Identity Inspector in Xcode, which is used for configuring UI components. It is divided into several sections:

- Custom Class:** Contains dropdown menus for 'Class' (set to 'UIView') and 'Module' (set to 'None'). There is also a checkbox for 'Inherit Module From Target'.
- Identity:** Includes a text field for 'Restoration ID'.
- User Defined Runtime Attributes:** A table with columns 'Key Path', 'Type', and 'Value'.
- Document:** Contains fields for 'Label' (set to 'Xcode Specific Label'), 'Object ID' (set to '8bC-Xf-vdC'), 'Lock' (set to 'Inherited - (Nothing)'), and 'Localizer Hint' (set to 'Comment For Localizer').
- Accessibility:** Includes a checkbox for 'Accessibility' (unchecked), and text fields for 'Label', 'Hint', and 'Identifier'. Below these are checkboxes for various traits: 'Button', 'Image', 'Static Text', 'Search Field', 'Plays Sound', 'Keyboard Key', 'Link', 'Selected', and 'Summary Element'.

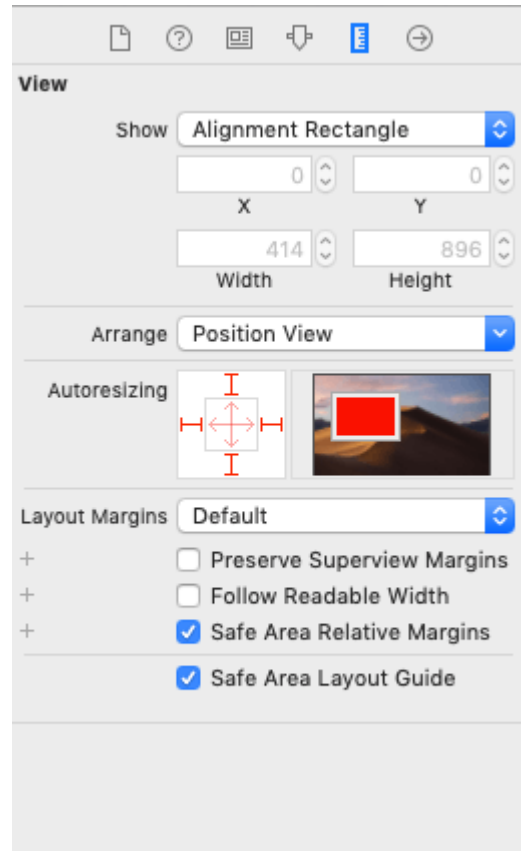
Inspector de atributos

Un inspector de atributos se utiliza para dar algunos atributos a la vista UIV correspondiente en el guión gráfico como modo de contenido, etiquetas, interacción, color de fondo, color de fuente, tamaño de fuente, etc. Los atributos otorgados a las vistas del guión gráfico usando el inspector de atributos son estáticos y pueden ser cambiado programáticamente en tiempo de ejecución.



Inspector de tallas

El inspector de tamaño proporciona información sobre las restricciones de tamaño que se le dan a la vista mientras se diseña usando un guión gráfico. Podemos alterar la restricción de tamaño dada a la vista usando el Inspector de tamaño.



Inspector de Conexiones

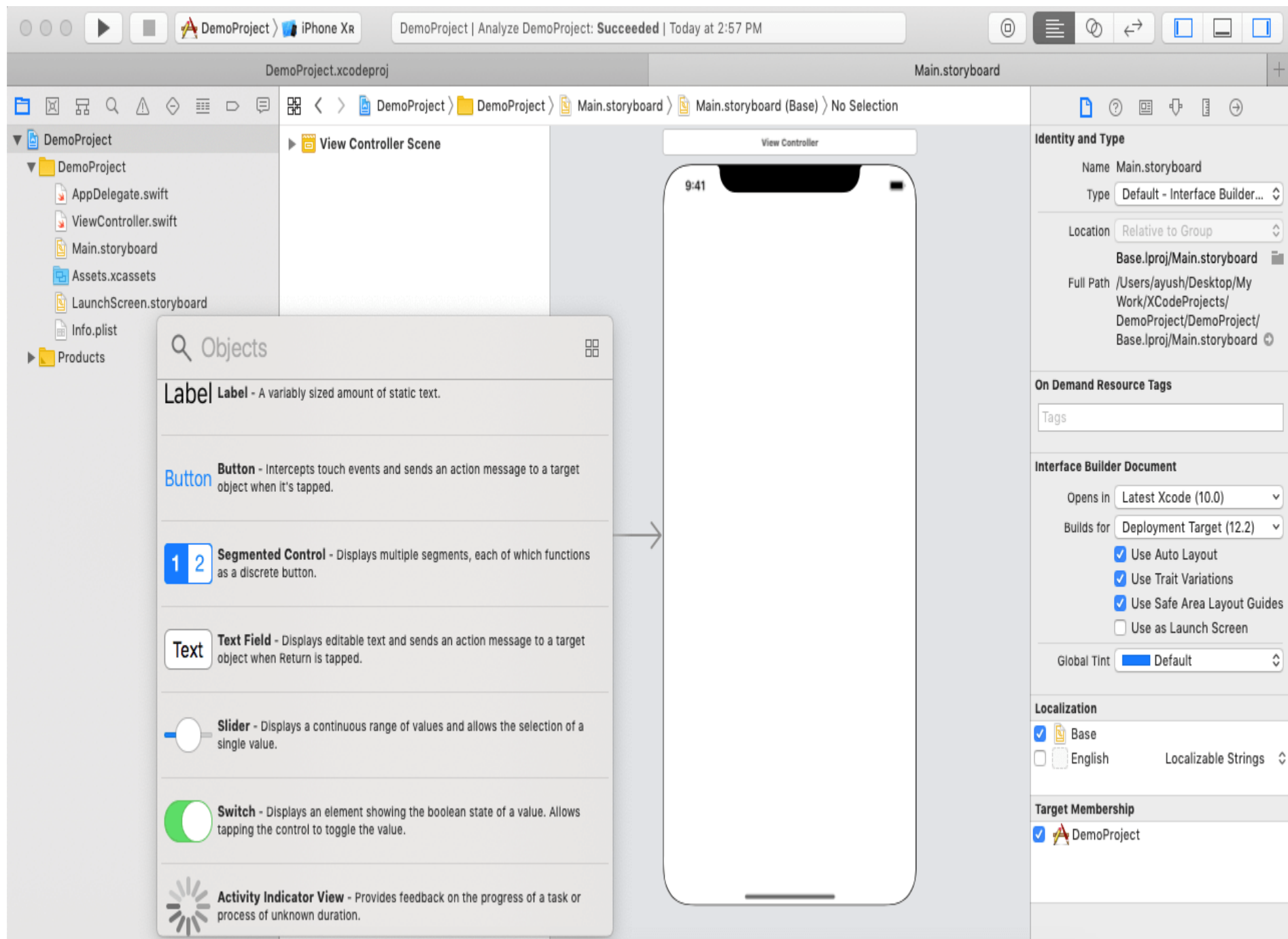
Muestra la información sobre las conexiones del guión gráfico correspondiente UIView al archivo de clase rápida. Contiene todas las conexiones del guión gráfico a los archivos de clase rápida.



Mediateca

vista de colección, etc., que se utiliza principalmente para crear una aplicación de iOS. Usando la biblioteca de medios, podemos insertar el widget deseado en el guión gráfico utilizando la funcionalidad de arrastrar y soltar. Podemos abrir la biblioteca de medios usando el **comando + shift + L** tecla corta.

En la parte inferior del Editor estándar, se muestra la consola del depurador, que se utiliza para depurar el programa en tiempo de ejecución. La consola también se usa para imprimir los valores mediante las funciones rápidas `print ()` y `debugPrint ()`.



iPhone Historia y Versiones

iPhone	Versión IOS	Fecha de lanzamiento	Precio de lanzamiento
iPhone	iPhone OS 1.0	29 de junio de 2007	\$ 499 / \$ 599
iPhone 3G	iPhone OS 2.0	11 de junio de 2008	\$ 199 / \$ 299 \$ 499
iphone 3gs	iPhone OS 3.0	19 de junio de 2009	\$ 199 / \$ 299 \$ 599 / \$ 699
Iphone 4	iPhone OS 4.0	21 de junio de 2010	\$ 199 / \$ 299 \$ 649 / \$ 749
iphone 4s	iPhone OS 5.0	14 de octubre de 2011	\$ 199 / \$ 299 / \$ 399 \$ 649 / \$ 749 / \$ 849
iphone 5	iPhone OS 6.0	21 de septiembre de 2012	\$ 199 / \$ 299 / \$ 399 \$ 649 / \$ 749 / \$ 849
iphone 5c	iPhone OS 7.0	20 de septiembre de 2013	\$ 99 / \$ 199 \$ 549 / \$ 649
iphone 5s	iPhone OS 7.0	20 de septiembre de 2013	\$ 199 / \$ 299 / \$ 399 \$ 649 / \$ 749 / \$ 849
IPhone 6/6 plus	iPhone OS 8.0	19 de septiembre de 2014	\$ 199 / \$ 299 / \$ 399 \$ 649 / \$ 749 / \$ 849 Plus: \$ 299 / \$ 399 / \$ 499 Plus: \$ 749 / \$ 849 / \$ 949
IPhone 6s / 6s plus	iPhone OS 9.0.1	25 de septiembre de 2015	\$ 199 / \$ 299 / \$ 399 \$ 649 / \$ 749 / \$ 849 Plus: \$ 299 / \$ 399 / \$ 499 Plus: \$ 749 / \$ 849 / \$ 949
IPhone SE	iPhone OS 9.3	31 de marzo de 2016	\$ 399 / \$ 499
IPhone 7/7 Plus	iPhone OS 10.0	16 de septiembre de 2016	\$ 199 / \$ 299 / \$ 399 \$ 649 / \$ 749 / \$ 849 Plus: \$ 319 / \$ 419 / \$ 519 Plus: \$ 769 / \$ 869 / \$ 969
IPhone 8/8 Plus	iPhone OS 11.0	22 de septiembre de 2017	\$ 699 / \$ 849 Plus: \$ 799 / \$ 949
IPhone X	iPhone OS 11.1	3 de noviembre de 20017	\$ 549 / \$ 699 \$ 999 / \$ 1149
IPhone XS / XS Max	iPhone OS 12.0	21 de septiembre de 2018	\$ 999 / \$ 1149 / \$ 1349 Máx .: \$ 1099 / \$ 1249 / \$ 1449
iPhone XR	iPhone OS 12.0	26 de octubre de 2018	\$ 749 / \$ 799 / \$ 899
iPhone 11	iPhone OS 13.0	20 de septiembre de 2019	\$ 699 / \$ 749 / \$ 849
IPhone 11 pro / iPhone 11 max	IPhone OS 13.0	20 de septiembre de 2019	\$ 999 / \$ 1149 / \$ 1349 Máx .: \$ 1099 / \$ 1249 / \$ 1449

Vistas y controladores de vista

En el desarrollo de iOS, los controladores de vista son la base de la estructura interna de la aplicación. El controlador de vista es el padre de todas las vistas presentes en un guión gráfico. Cada aplicación tiene al menos un ViewController. Facilita la transición entre varias partes de la interfaz de usuario.

El UINavigationController es la clase principal de todos los ViewControllers. Define todos los métodos y propiedades para administrar nuestras vistas. Esta clase también gestiona los eventos y las transiciones de un controlador de vista a otro. También coordina entre las diferentes partes de la aplicación.

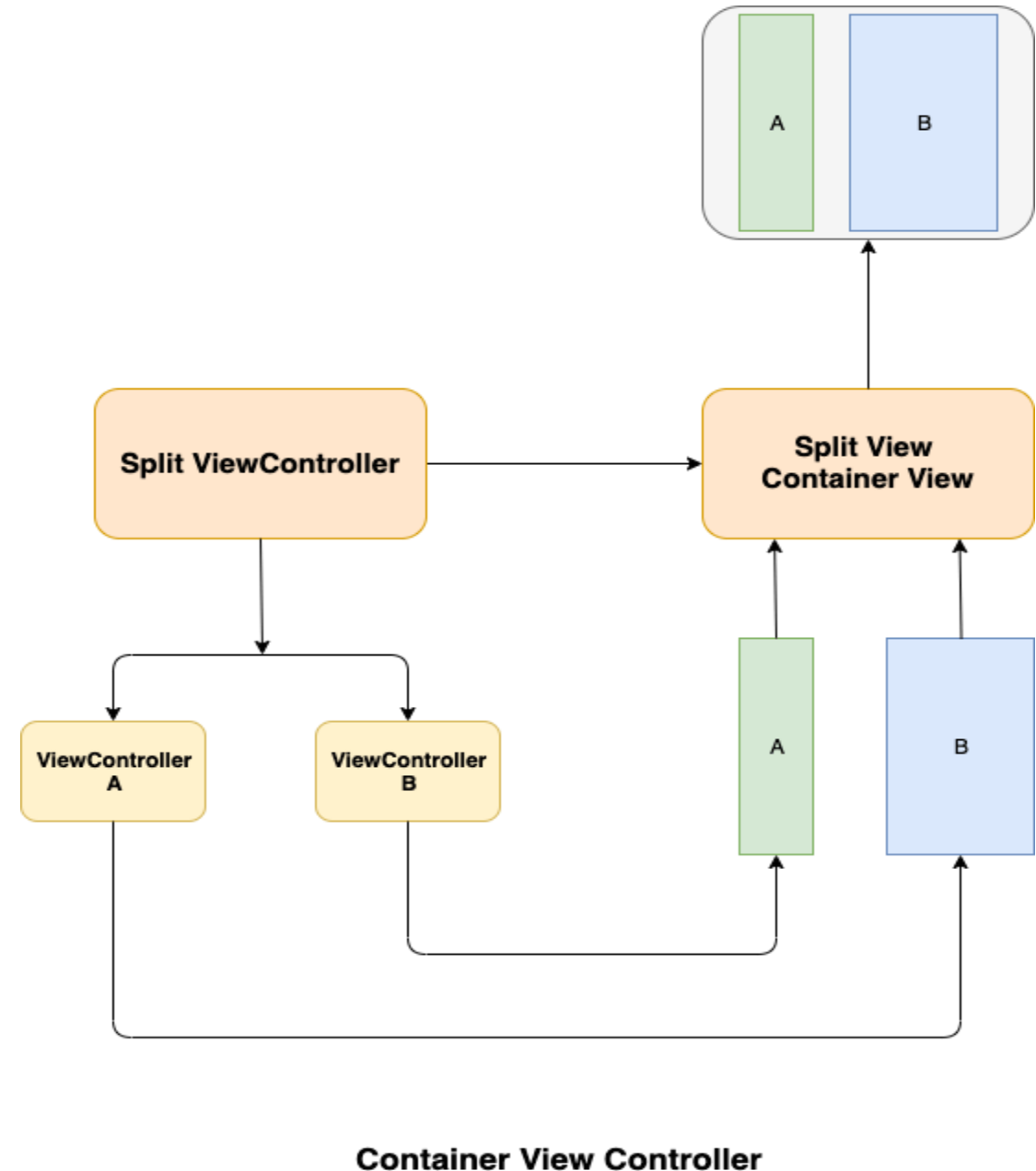
Tipos de controladores de vista

Hay dos tipos de ViewControllers:

1.Content ViewController: Content ViewControllers es el tipo principal de View Controllers que creamos. Content View Controllers contiene el contenido de la pantalla de la aplicación. En otras palabras, podemos decir que el Controlador de vista de contenido administra la parte discreta del contenido de la aplicación. Content ViewController gestiona todas las vistas en sí.

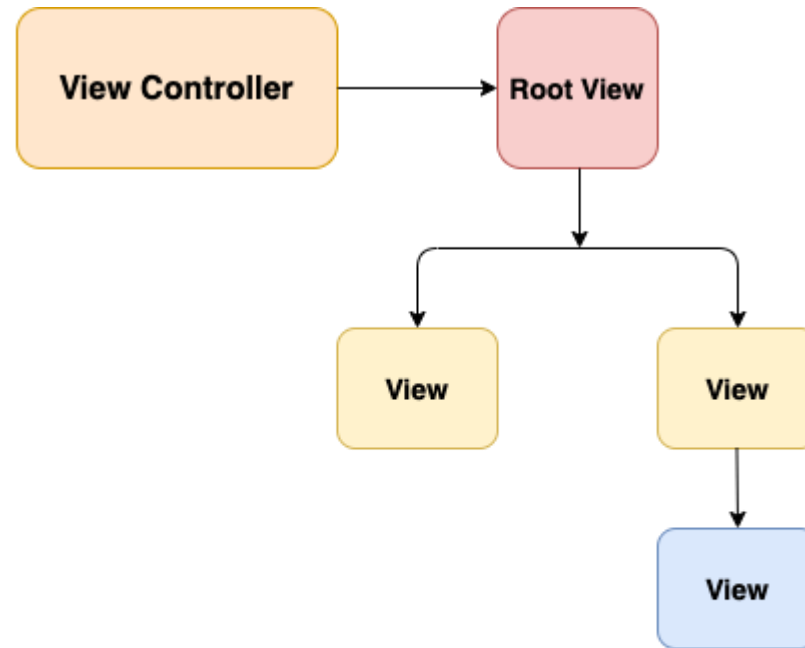
2.Container ViewController: Container ViewController es diferente del contenido ViewController en el sentido de que actúa como un controlador de vista principal, que recopila información de los controladores de vista secundarios. La tarea del controlador de vista de contenedor es presentar la información recopilada para facilitar la navegación a los controladores de vista secundarios. El contenedor ViewController solo administra RootView, que incorpora uno o más Child ViewControllers.

La mayoría de las aplicaciones de iOS son la mezcla de ambos, Content ViewController y Container ViewController.



Ver gestión

En el desarrollo de iOS, ViewController gestiona la jerarquía de vistas. Como se muestra en la imagen a continuación, cada ViewController contiene un rootView que contiene todo el contenido del controlador de vista. Todas las vistas personalizadas necesarias para mantener una aplicación iOS se agregan a la vista raíz para mostrar el contenido. La siguiente figura muestra la relación entre ViewController, rootView y sus subvistas. Cada vista secundaria es referida por una Súper Vista que incorpora una cadena de Vistas donde un rootView actúa como la Vista Principal de todas las vistas presentes en los Controladores de Vista.

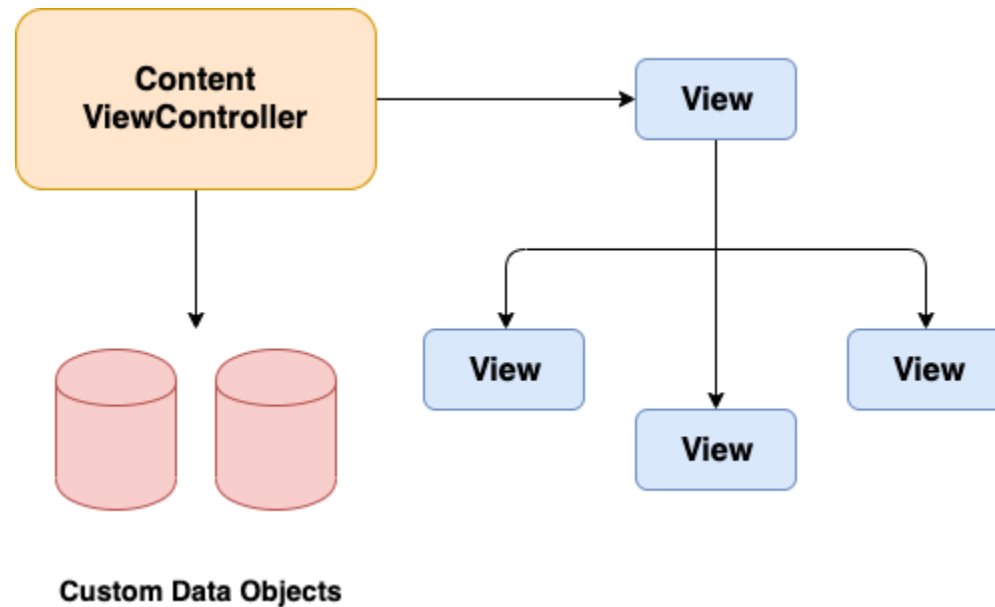


Relationship between a ViewController and its Views

Data Marshaling

En iOS Development, un controlador de vista es responsable de mostrar los datos de nuestra aplicación iOS en la pantalla. Actúa como una interfaz entre sus Vistas (creadas por el desarrollador) y los datos de la aplicación. A cada ViewController en el Storyboard se le asigna una Clase que hereda la clase UIViewController.

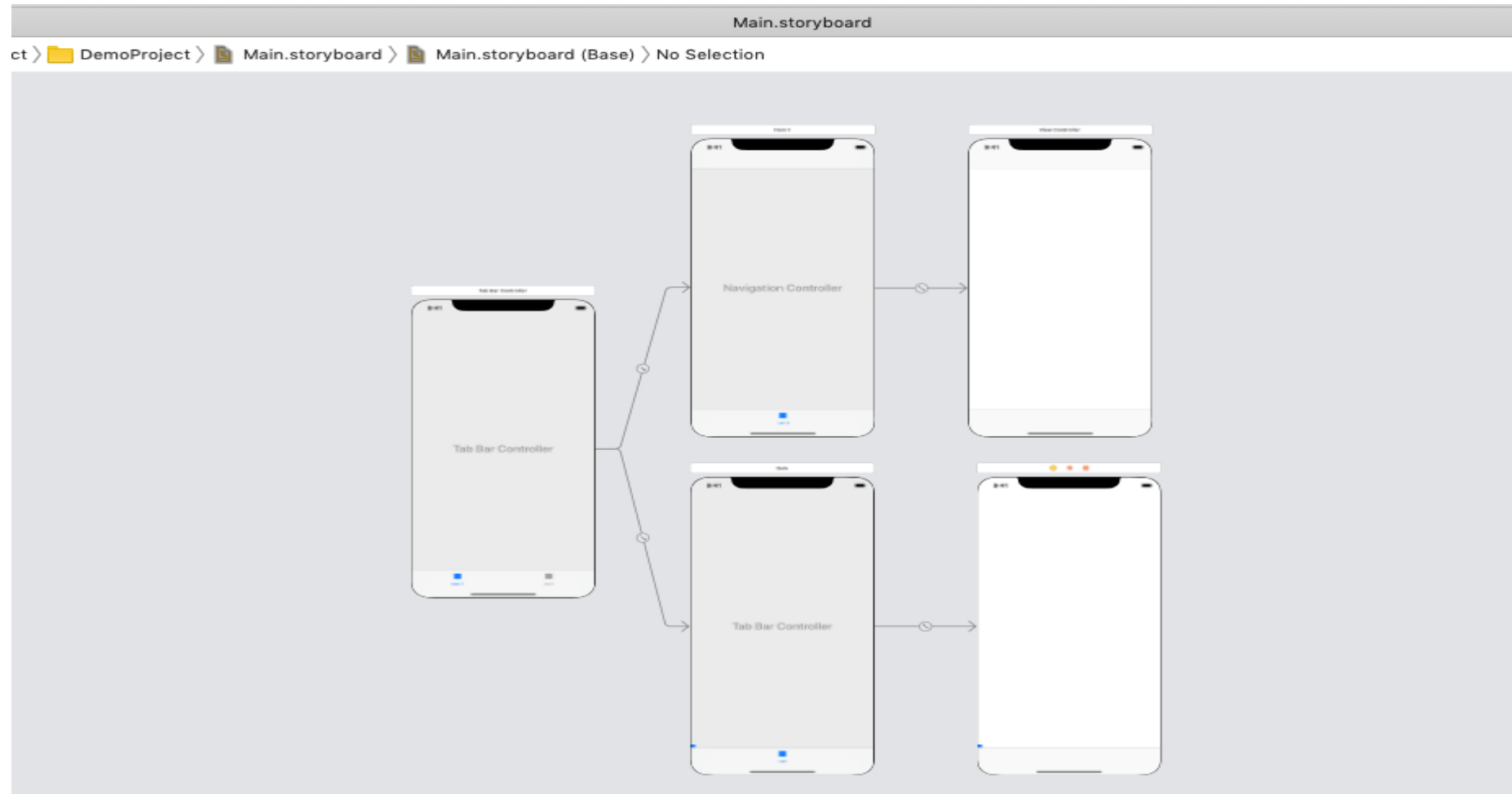
Todas las propiedades y métodos definidos en el UIViewController están presentes en la clase que asignamos al ViewController. Sin embargo, para el desarrollo de nuestra aplicación, necesitamos definir nuestras propiedades y métodos en la clase ViewController. Nos ayuda a gestionar la representación visual de nuestra aplicación.



Storyboard and Interface Builder

El gui3n gr3fico se introdujo por primera vez en iOS 5 para ahorrar tiempo al crear interfaces de usuario para las aplicaciones de iOS. Es una representaci3n visual de la interfaz de usuario de una aplicaci3n de iOS. Se puede definir como la secuencia de pantallas, cada una de las cuales representa el ViewController y las Vistas. Las transiciones entre dos pantallas de storyboard necesitan un objeto segue, que representa una transici3n entre dos ViewControllers.

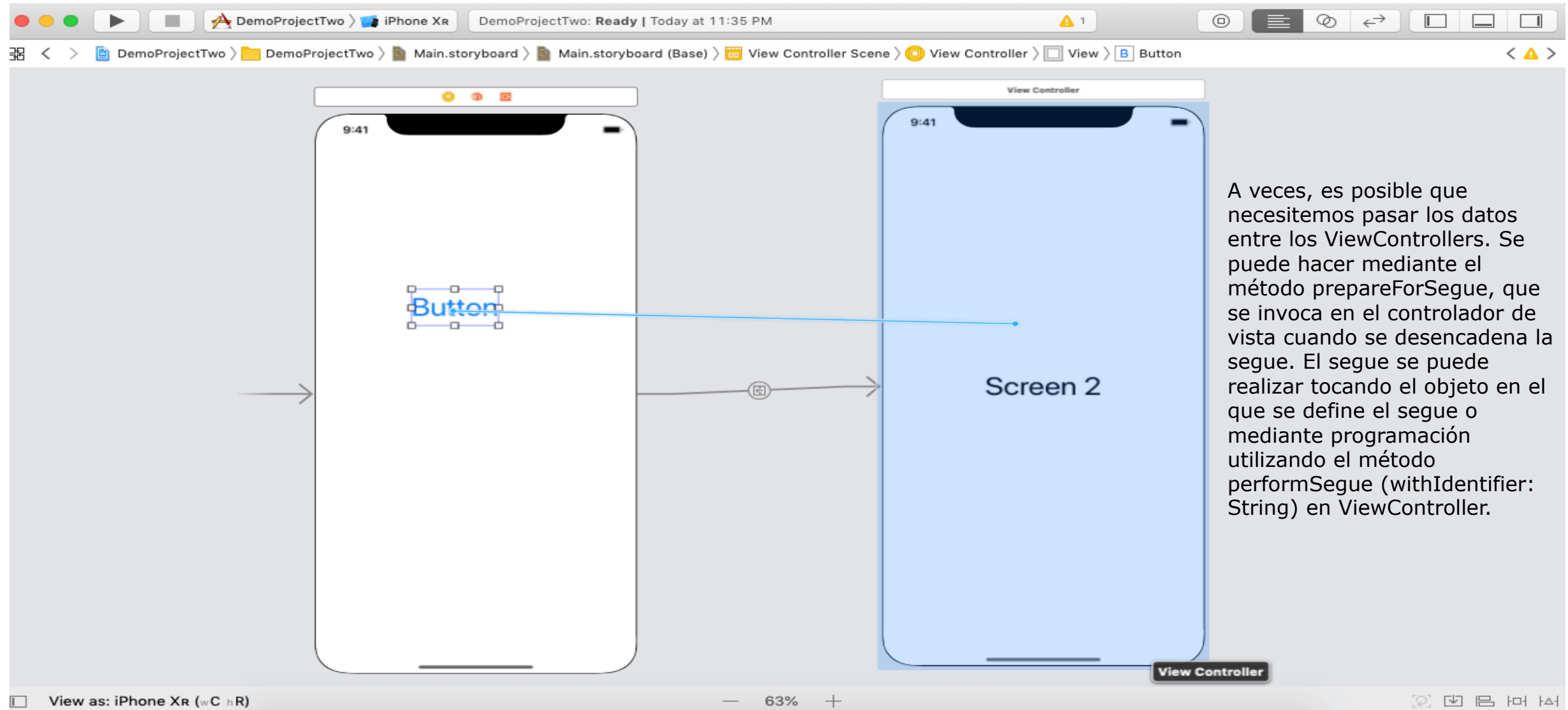
El Storyboard se construye utilizando un editor visual proporcionado por XCode, en el que podemos dise1ar y dise1ar las interfaces de usuario de la aplicaci3n agregando los widgets de la biblioteca de medios, como botones, vistas, vistas de tabla, campos de texto, etc. Todas las vistas ser3 cubierto en detalle m3s adelante en este tutorial.



Segues

Los segmentos se utilizan para hacer las transiciones entre dos pantallas en el guión gráfico. Podemos establecer el tipo de transición como el modelo o seguir adelante. En palabras simples, el segue es como una flecha definida en un objeto como un botón o ViewController para que cualquier evento de usuario en el objeto conduzca a la transición definida por el segue.

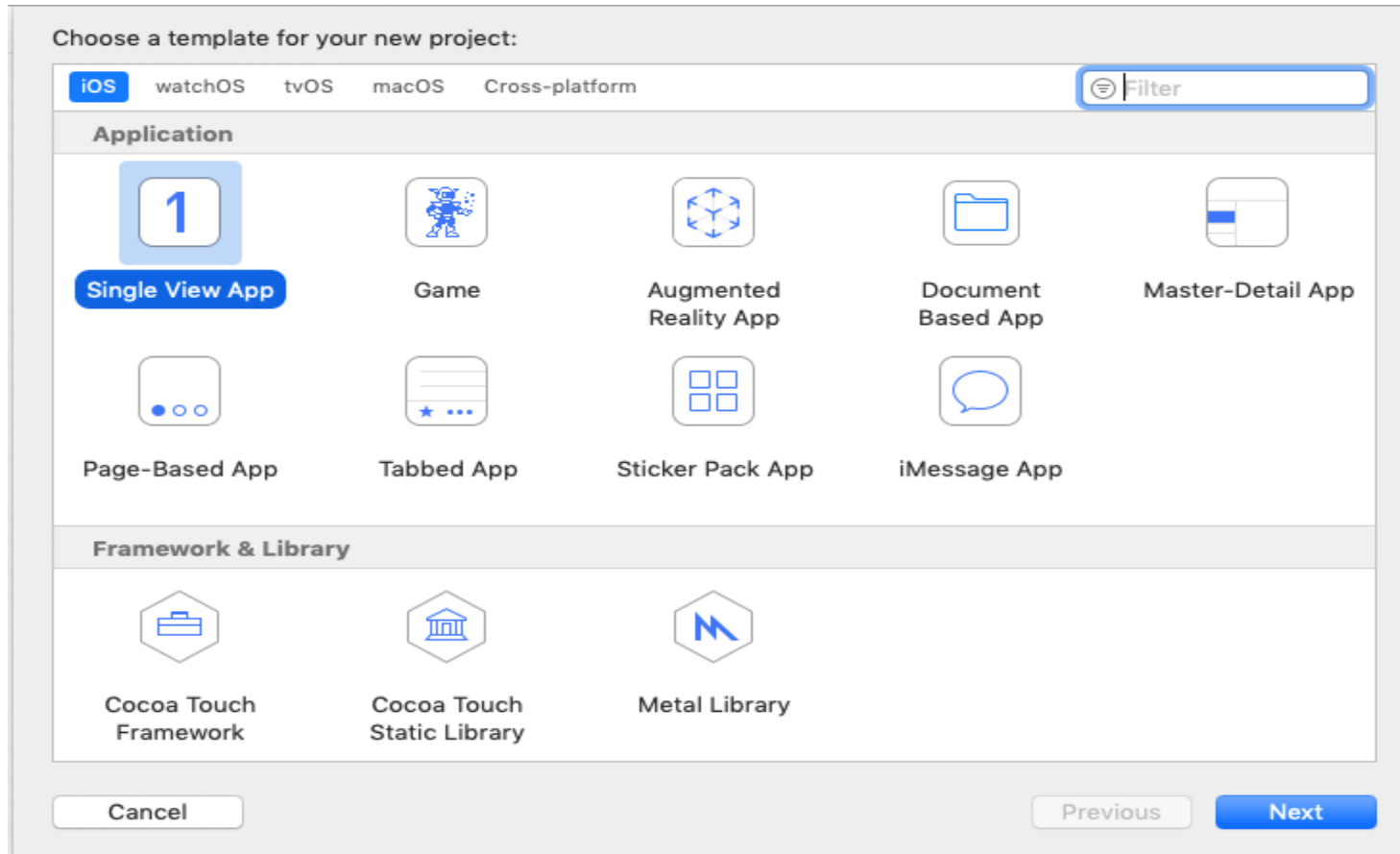
La siguiente imagen muestra la transición entre dos ViewControllers usando Segue.



Crear la primera aplicación de iOS

Hasta ahora, hemos visto los conceptos básicos de cómo se puede desarrollar una aplicación iOS usando XCode en macOS. También hemos discutido sobre la necesidad del generador de interfaces, los controladores de vista y el guión gráfico. En esta sección del tutorial, crearemos nuestra primera aplicación para iOS y discutiremos algunos de los componentes básicos de una aplicación para iOS.

Para crear la primera aplicación de iOS, necesitamos crear un nuevo proyecto en XCode haciendo doble clic en el icono XCode en las aplicaciones. Esto abrirá el XCode, debemos elegir la opción Crear un nuevo proyecto en el panel izquierdo de la ventana. Se abrirá la siguiente ventana. XCode nos proporciona varias opciones de tipo de proyecto para elegir. Aquí, debemos seleccionar la aplicación Single View. Discutiremos otros tipos de proyectos también, más adelante en este tutorial.



Ahora haga clic en Siguiente. Aparecerá la siguiente ventana, que nos solicita que completemos la información relacionada con el proyecto. Complete todos los detalles sobre el producto, incluido el nombre, el equipo, el nombre de la organización, etc. Aquí, debemos notar que la selección predeterminada para el equipo es Ninguno. Aquí, podemos seleccionar el nombre del desarrollador, para esto, tenemos que iniciar sesión con la ID de Apple. Sin embargo, lo he mantenido como Ninguno.

En Nombre de la organización, podemos completar el nombre para el que estamos desarrollando la aplicación. Ingresé a Apple Inc para este proyecto.

En el identificador de la organización, necesitamos completar el identificador que es inverso al nombre de dominio de la organización. Por ejemplo, para la organización Javatpoint, la ingresaremos como com.javatpoint

El identificador de paquete se creará concatenando el identificador de la organización y el nombre del producto. El identificador de paquete es el identificador único que representará la aplicación en una plataforma específica como firebase. No podemos instalar dos aplicaciones iOS en un iPhone con el mismo identificador de paquete.

En el idioma, seleccione el idioma con el que estamos planeando el desarrollo. En este tutorial, usaremos Swift como nuestro lenguaje de programación.

Choose options for your new project:

Product Name: DemoProject

Team: None

Organization Name: Apple Inc

Organization Identifier: com

Bundle Identifier: com.DemoProject

Language: Swift

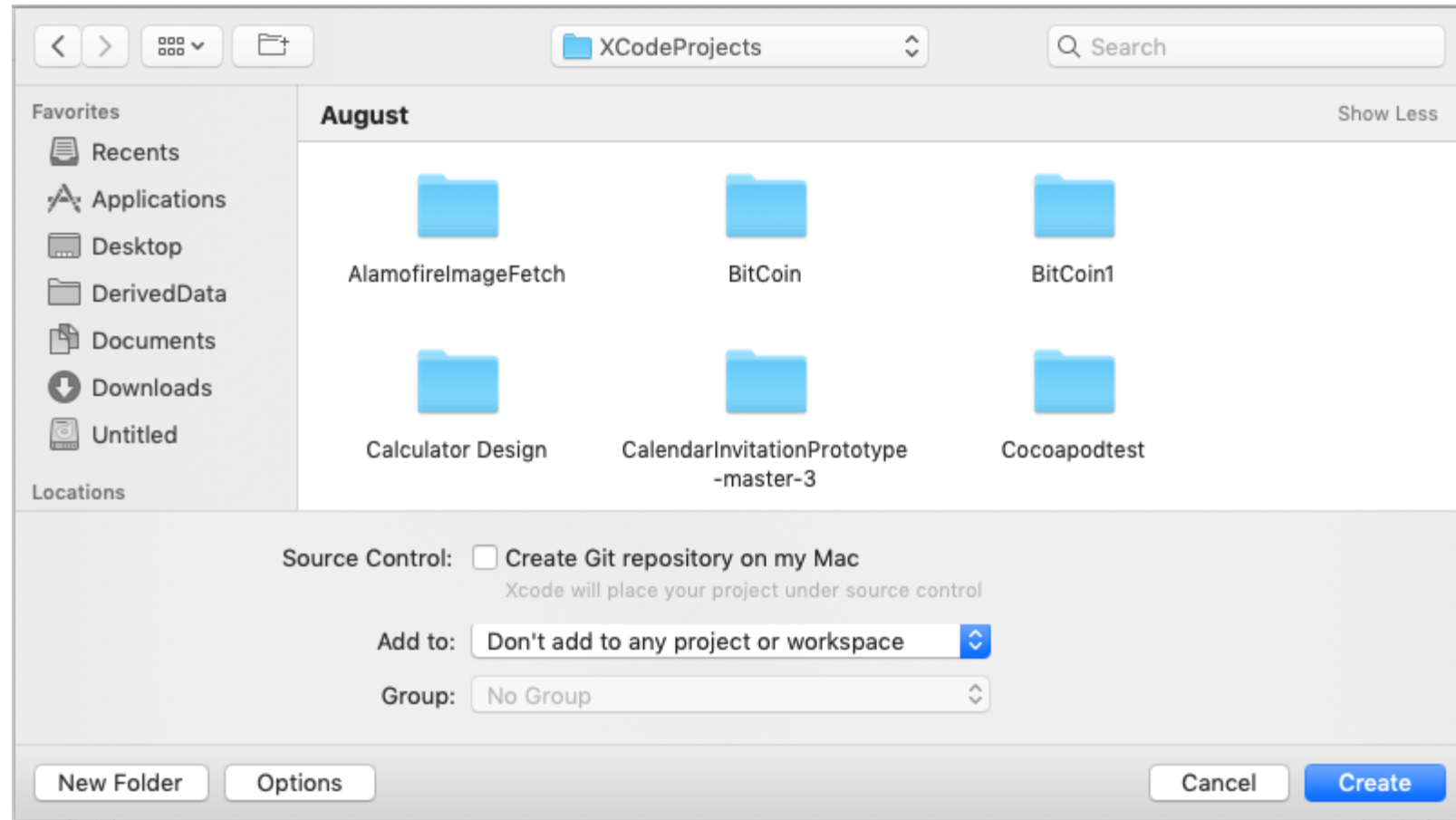
☐ Use Core Data

☐ Include Unit Tests

☐ Include UI Tests

Cancel Previous Next

En la parte inferior, se nos solicita que verifiquemos tres opciones. Inicialmente, no seleccionaremos ninguno de ellos. Sin embargo, las opciones son necesarias cuando usaremos CoreData en aplicaciones iOS.



Ahora, nos pedirá que seleccionemos la ubicación en el sistema para guardar el proyecto. Seleccione la ubicación en la que desea guardar el proyecto y haga clic en Crear. Sin embargo, también nos pedirá que seleccionemos la herramienta de control de versiones que es si necesitamos mantener el repositorio de Git en nuestro sistema. Seguiremos sin marcar y haremos clic en Crear. Esto creará con éxito nuestra primera aplicación para iOS. Ahora, programemos nuestra primera aplicación de iOS para mostrar un mensaje al usuario en la pantalla. Comenzaremos con la aplicación Hello World para iOS.

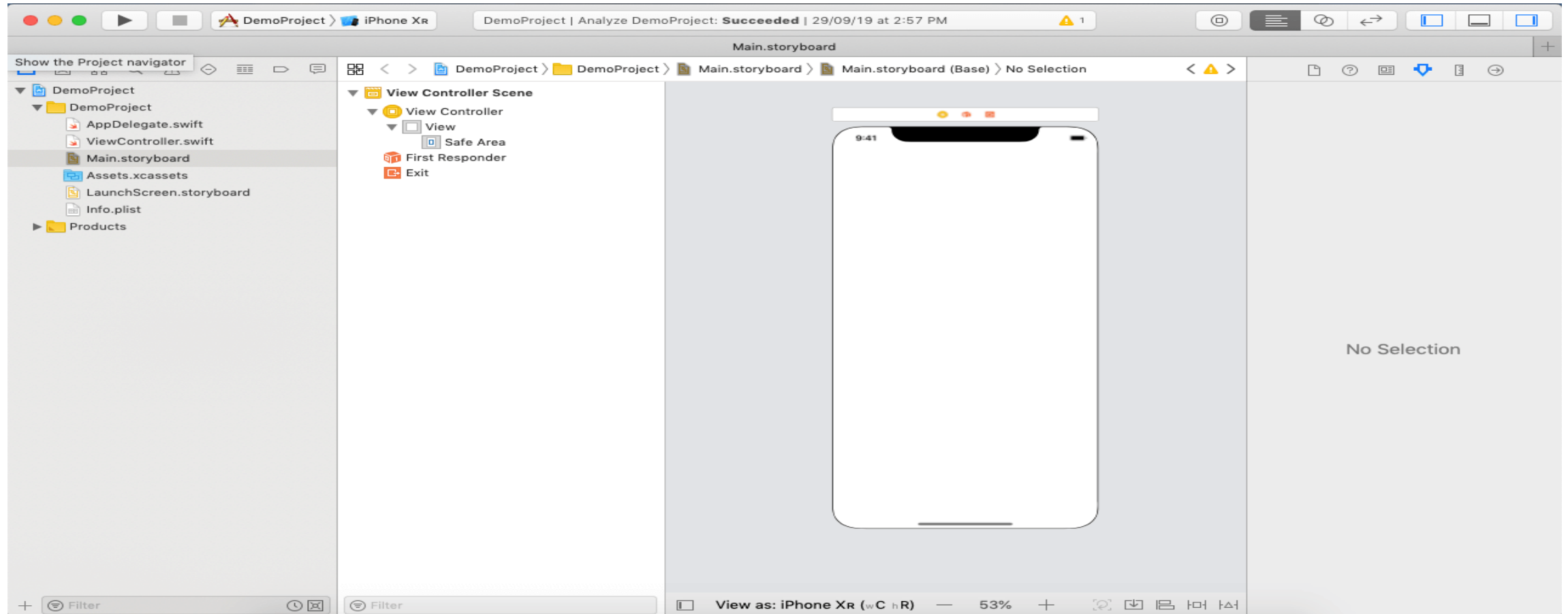
Primera aplicación de iOS

Nuestra primera aplicación iOS mostrará un primer mensaje decorado y popular para el usuario de iPhone, es decir, Hello World.

Crearemos esta aplicación utilizando un guión gráfico. Después de crear esta aplicación, obtendrá una idea de cómo puede usar storyboard de manera eficiente para crear una aplicación iOS. En este proyecto, para una mejor comprensión, no aplicaremos las restricciones a las Vistas, ya que se tratarán en la parte posterior de este tutorial, es decir, Diseño automático en forma rápida.

Cuando abrimos el guión gráfico principal desde el navegador del proyecto, se verá como la siguiente ventana. Ahora, podemos arrastrar y soltar los widgets desde la biblioteca de medios.

Como hemos discutido anteriormente en este tutorial, los ViewControllers son los componentes básicos de cualquier aplicación de iOS. Inicialmente, cualquier proyecto de iOS está asociado con un controlador de vista en el guión gráfico y una clase asignada al controlador de vista, es decir, ViewController.swift. Podemos agregar The UIViews en este ViewController y programar las Vistas en el archivo de clase correspondiente creando los puntos de venta.

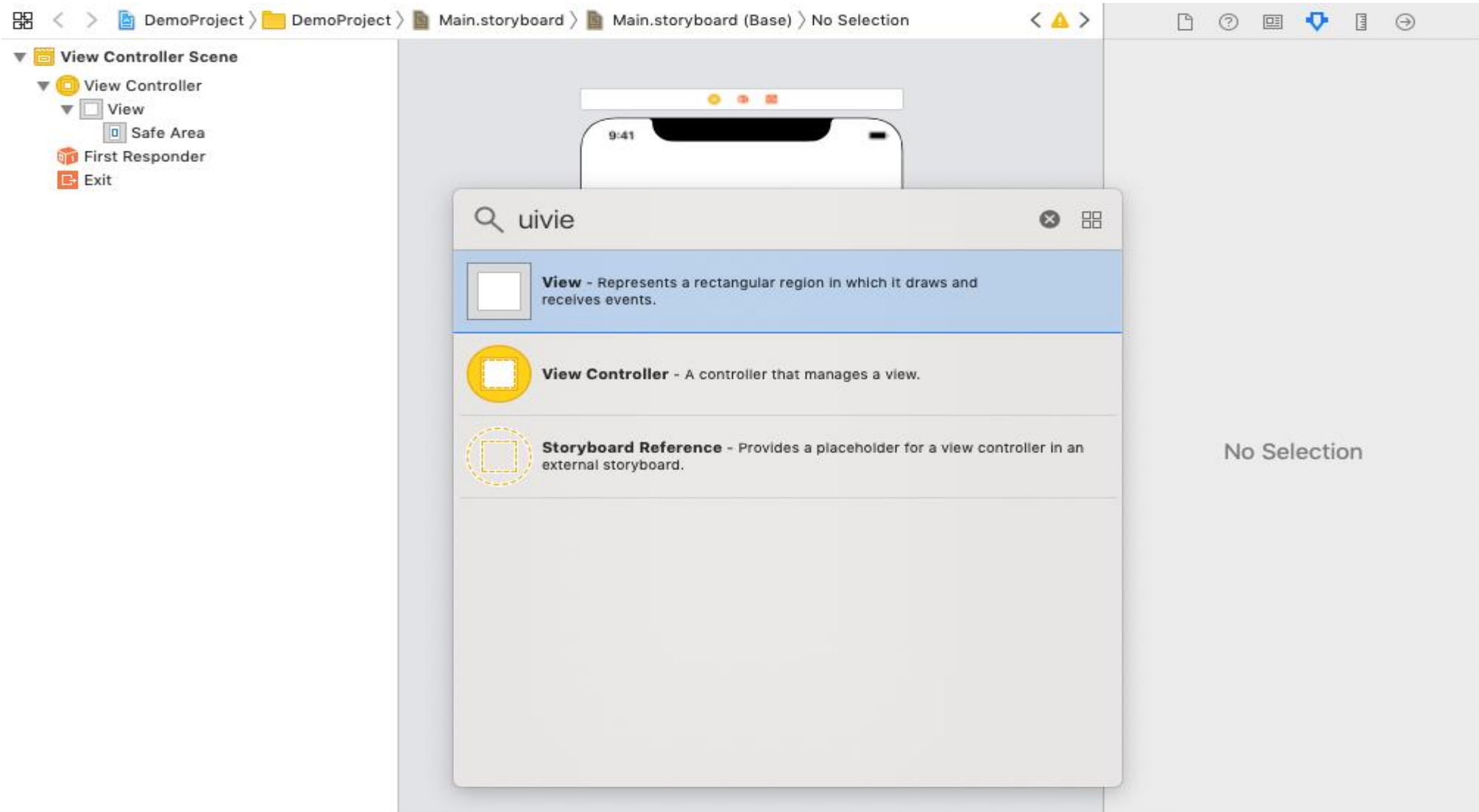


Si observamos los requisitos de esta aplicación, debemos mostrar un mensaje decorado Hello World al usuario de iOS.

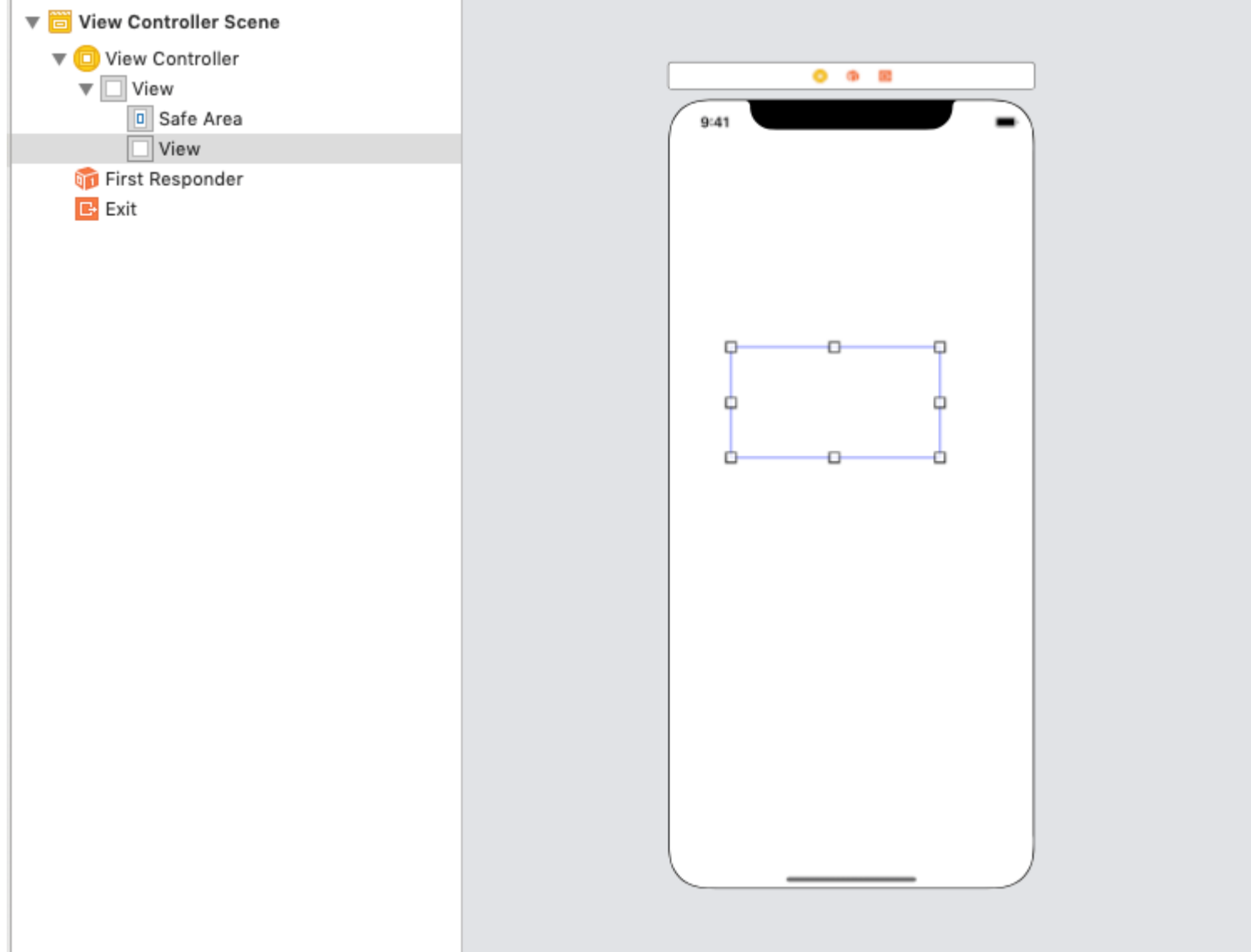
Para este propósito, seguiremos los siguientes pasos.

Paso 1: agregue un UIView al ViewController en el guión gráfico.

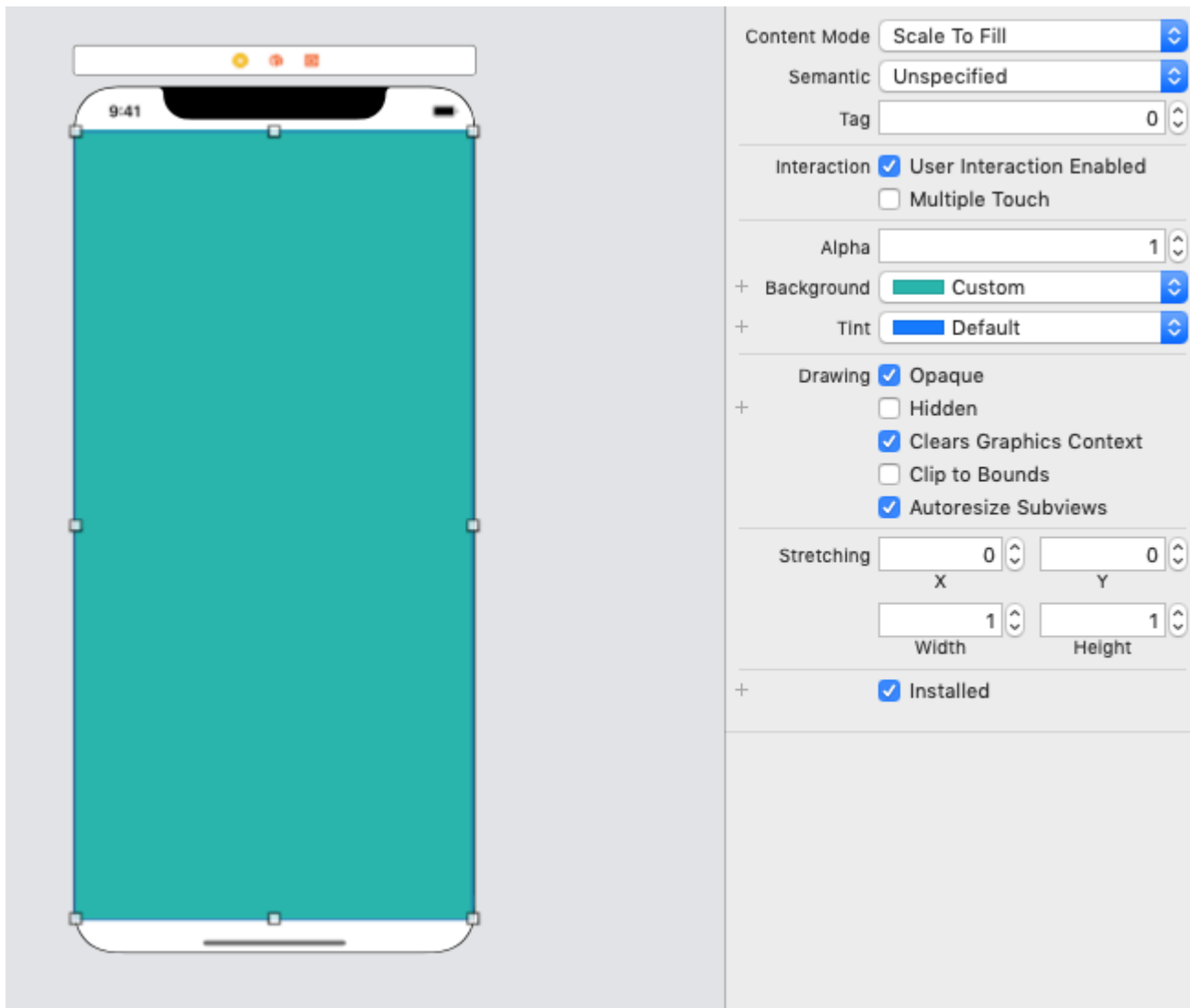
Para agregar una UIView al ViewController, abra la biblioteca de medios presionando la tecla corta Comando + Mayús + L. Busque UIView y arrastre el resultado al guión gráfico en ViewController.



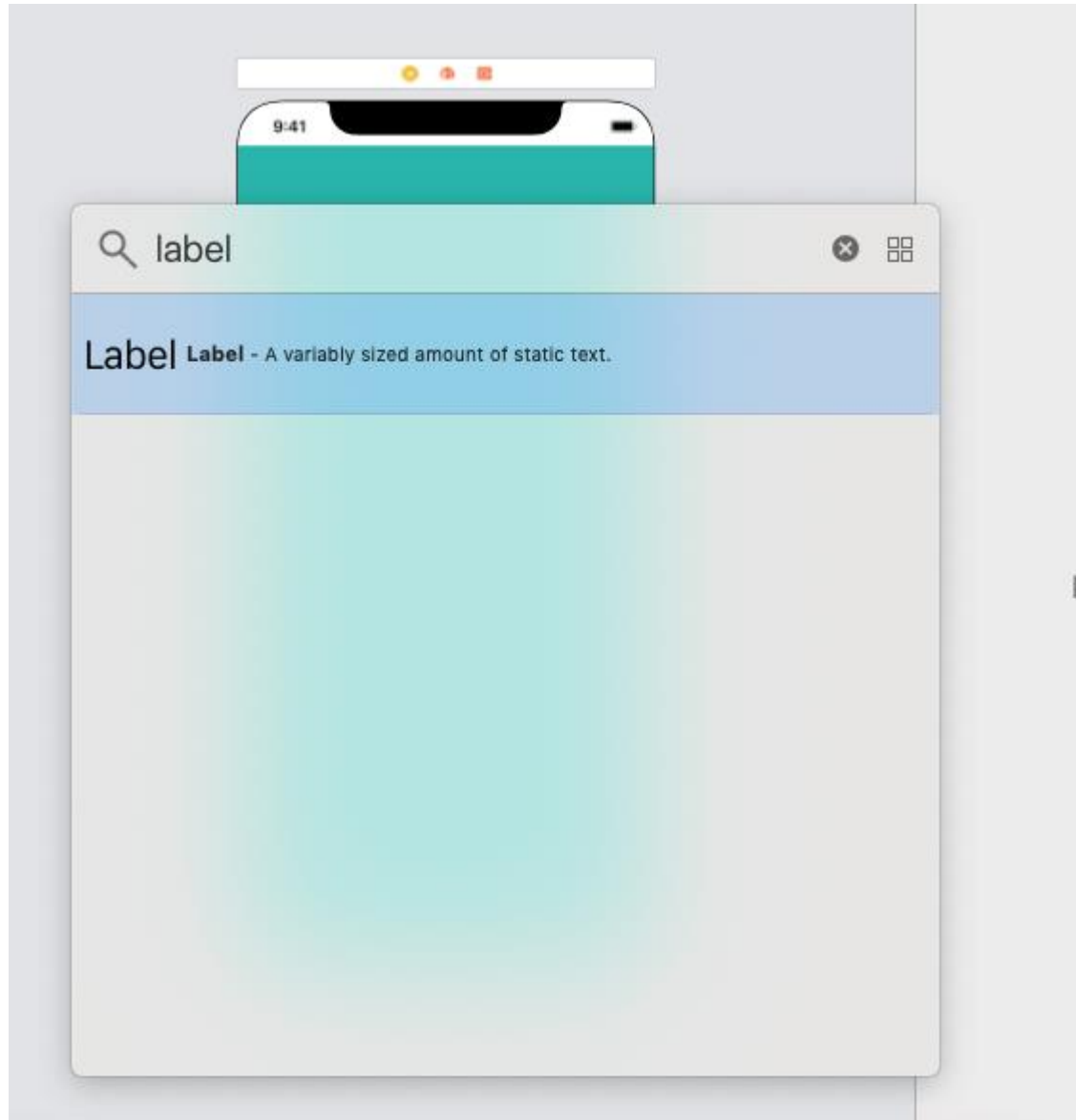
Esto agregará una UIView al ViewController en el storyboard



Ahora decore la UIView asignando un color y tamaño de fondo para la vista. Esto se hará utilizando el inspector de atributos en XCode, como se muestra en el panel derecho de la siguiente ventana.



Paso 2: Agregue un UILabel al ViewController en el guión gráfico. Ahora agregue un UILabel al ViewController en el guión gráfico de la biblioteca de medios. Busque UILabel en la biblioteca de medios y arrastre el resultado al Guión gráfico.

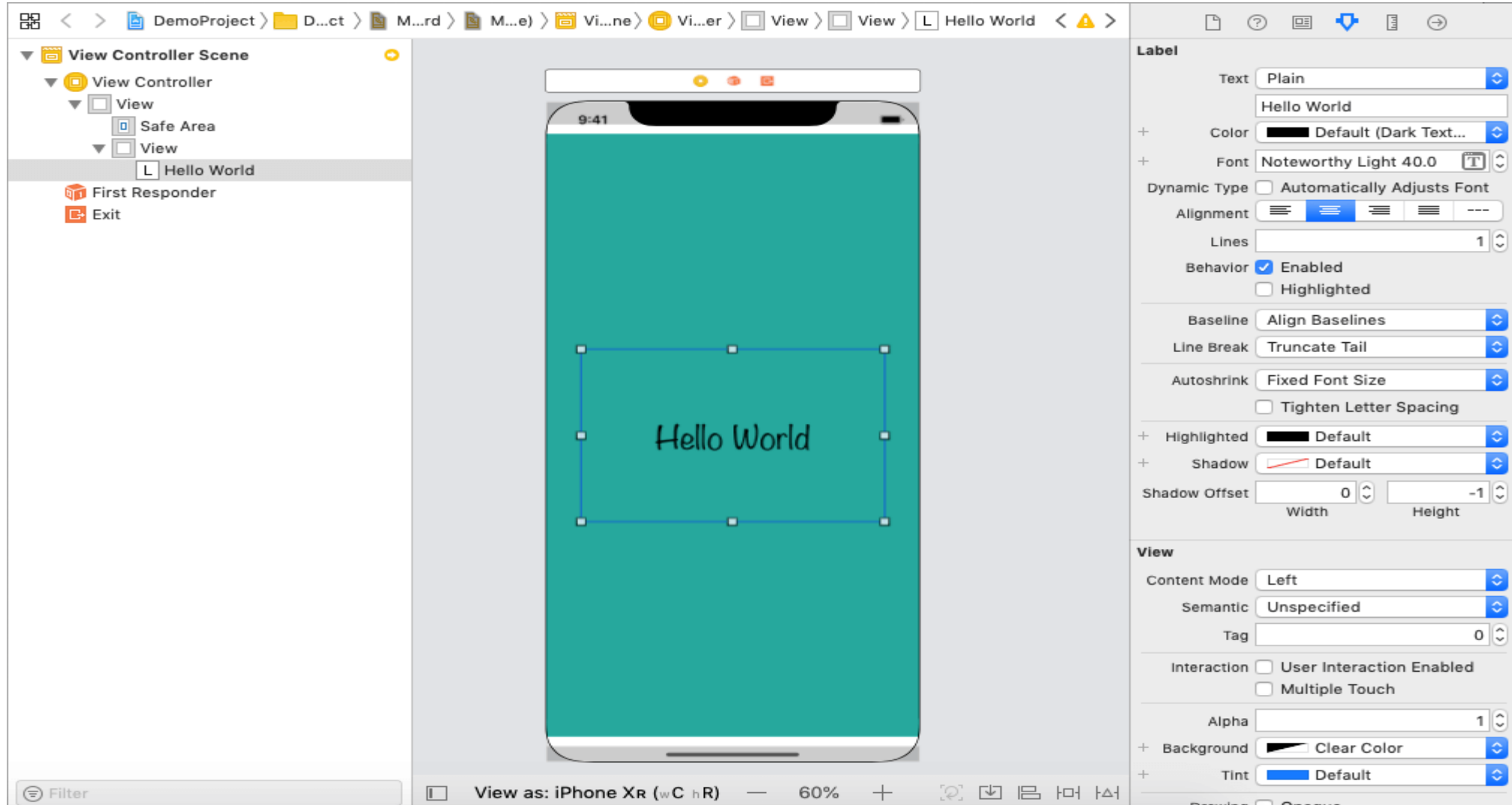


Esto agregará una etiqueta al Guión gráfico.

Paso 3: Decora el UILabel.

Ahora, decoraremos el UILabel dándole algunos atributos del inspector de atributos. En este proyecto, utilizaremos el mismo color de fondo que la Vista. Asignaremos una fuente personalizada a la etiqueta y colocaremos la etiqueta en el centro. También cambiaremos el texto de UILabel a Hello World como se muestra en la siguiente ventana.

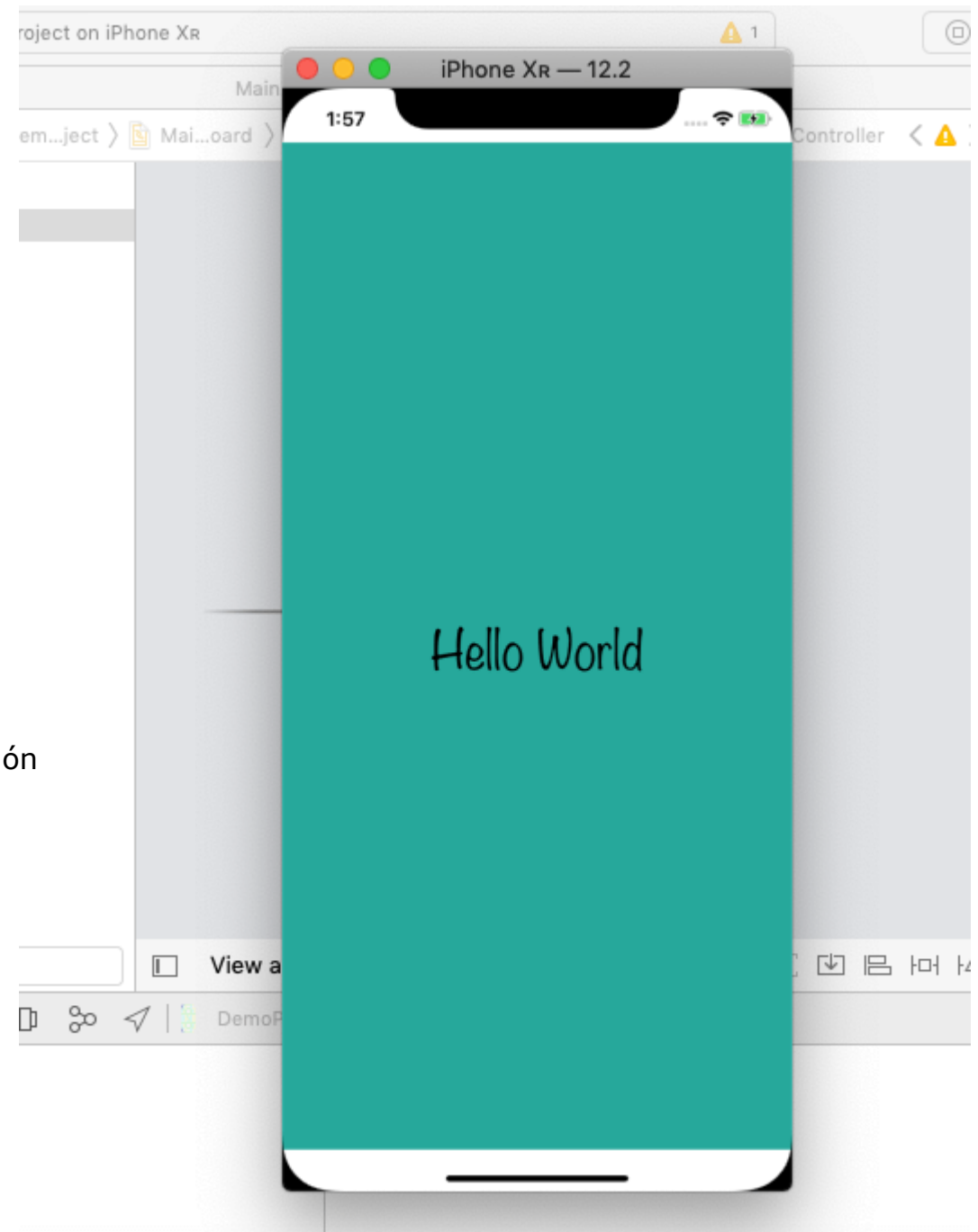
Los atributos otorgados a UILabel se pueden mostrar en el inspector de atributos. La escena del controlador de vista se muestra en el panel izquierdo de la ventana que muestra la jerarquía de vistas utilizadas en el guión gráfico.



Para ejecutar el proyecto, use la tecla corta Comando + R, esto abrirá el simulador para ejecutar el proyecto. El simulador simula el iPhone en macOS. Hay una amplia gama de simuladores disponibles en XCode para ejecutar la aplicación iOS en macOS. Por defecto, iPhone XR está seleccionado en el XCode para ejecutar la aplicación iOS. Ejecutemos la aplicación anterior en el iPhone XR.

Por lo tanto, hemos creado y ejecutado con éxito nuestra primera aplicación de iOS usando Simulator.

[Haga clic aquí para descargar el proyecto](#)



Controles de IU de iOS

Label

UILabel hereda la clase UIView. Representa una clase de vistas que muestran una o más líneas de textos de solo lectura. En las aplicaciones de iOS, la etiqueta se utiliza en asociación con UIControls para cumplir con los requisitos de la aplicación.

La sintaxis de la clase UILabel se da de la siguiente manera.

```
class UILabel : UIView
```

La apariencia de la etiqueta se puede configurar. Podemos personalizar las subcadenas dentro de la etiqueta. La etiqueta también puede mostrar cadenas atribuidas. Podemos agregar las etiquetas a la interfaz mediante programación o mediante un guión gráfico. Los siguientes pasos se utilizan para agregar la etiqueta a la interfaz.

1. Busque la etiqueta en la biblioteca de objetos y arrastre el resultado al guión gráfico o cree un objeto de la clase UILabel en la clase ViewController.
2. Configure la apariencia de la etiqueta utilizando el inspector de atributos.
3. Configure reglas de diseño automático para definir el tamaño y la posición de la etiqueta en su interfaz.

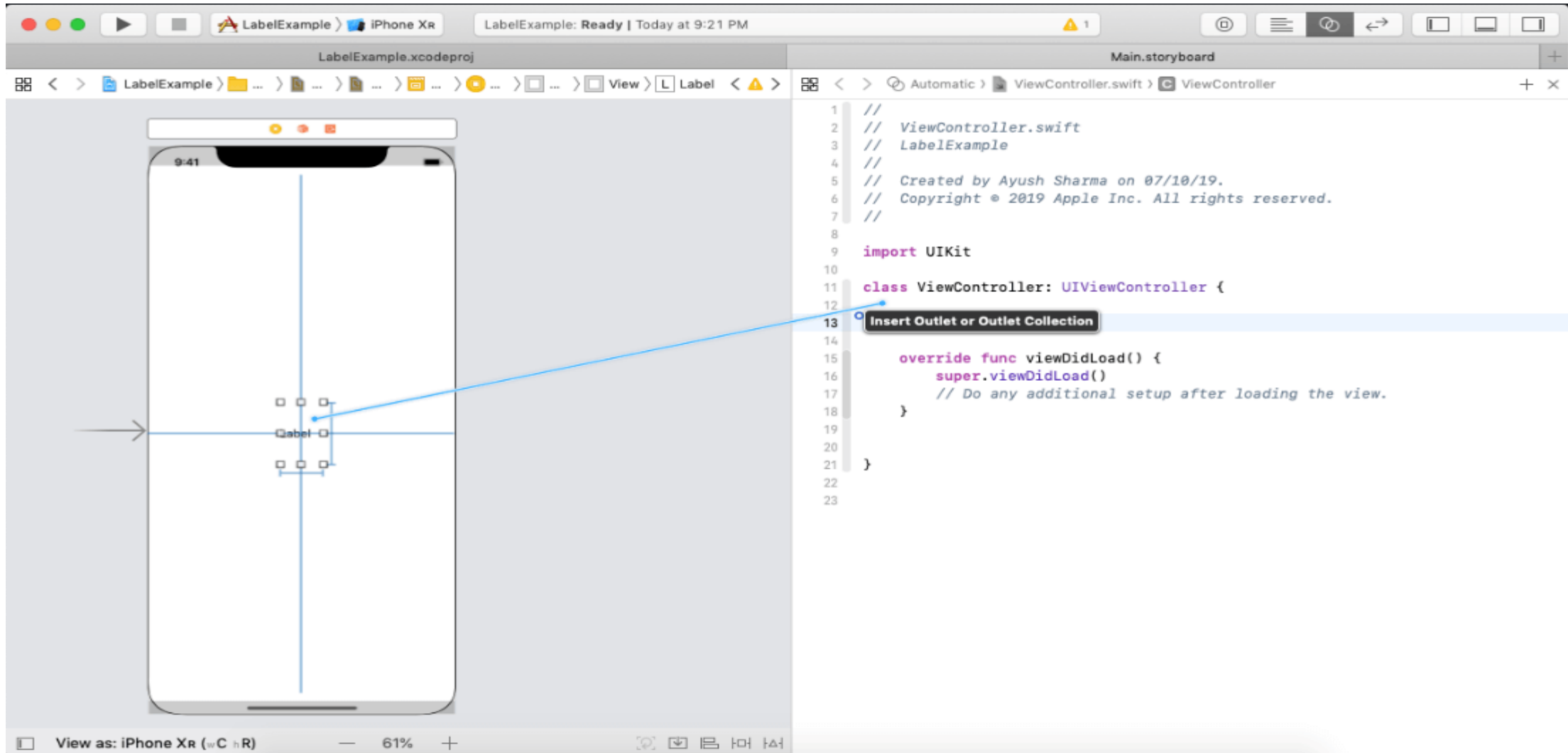
Atributos del generador de interfaz para UILabel

1	Text	Este atributo se puede dar para establecer el contenido de la etiqueta. Para definir la apariencia de la etiqueta, podemos asignar dos modos a la etiqueta, es decir, el modo normal y el modo atribuido. El modo simple muestra el contenido de la etiqueta con la apariencia uniforme, mientras que el modo atribuido aplica atributos de estilo dentro de la cadena. En modo atribuido, podemos usar el menú Más para revelar atributos de apariencia adicionales. Se puede acceder a este valor en tiempo de ejecución con el texto y las propiedades de texto atribuido.
2	Color	Se utiliza para establecer el color de fuente de la etiqueta. En modo plano, el color se establece para todo el contenido, mientras que, en modo atribuido, podemos definir el color para una parte particular del contenido.
3	Font	Esto se utiliza para alterar la fuente del contenido de la etiqueta. Incluye la familia de fuentes, el tamaño y la opacidad del contenido. Podemos definir el sistema o la fuente personalizada a la etiqueta. En la fuente personalizada, podemos establecer la familia de fuentes y el estilo del contenido de la etiqueta.
4 4	Alignment	Se utiliza para establecer la alineación del contenido de la etiqueta dentro del marco. Podemos colocar el contenido de la etiqueta, alinear a la izquierda, centro, alinear a la derecha, justificado o natural dentro del marco. En modo plano, la alineación se establece para todo el contenido de la etiqueta, mientras que, en modo atribuido, podemos mantener la alineación específica para el párrafo específico de la etiqueta.
5 5	Lines	Representa el número máximo de líneas que la etiqueta usa para representar el contenido. Podemos establecerlo en 0 para que la etiqueta represente las líneas ilimitadas. Se puede acceder a este valor en tiempo de ejecución mediante el uso de la propiedad numberOfLines en el objeto de etiqueta.
6 6	Behavior	Este atributo se usa para controlar el comportamiento de la etiqueta. Habilitado y resaltado controlan la apariencia de la etiqueta. Se puede acceder a estos valores en tiempo de ejecución utilizando las propiedades isEnabled y isHighlighted, respectivamente.
7 7	Baseline	Es un atributo de espaciado que controla la alineación vertical de la etiqueta cuando el autoencogimiento está habilitado. Se puede acceder a este valor utilizando la propiedad baselineAdjustment.
8	LineBreaks	Especifica el comportamiento de la etiqueta cuando el contenido es demasiado grande para ajustarse dentro de los límites de la etiqueta. Se puede establecer en ajuste de palabras o ajuste de caracteres para dividir el contenido en varias líneas por palabras o por caracteres. Se puede acceder a esta propiedad utilizando la propiedad lineBreakMode en el objeto de etiqueta.
9 9	Autoshrink	Este atributo se usa para alterar el tamaño de fuente del contenido de la etiqueta antes de recurrir al truncamiento. Se puede establecer en una escala de fuente mínima o en un tamaño de fuente máximo. Podemos elegir la escala de fuente mínima y establecer el valor para permitir que la etiqueta reduzca el tamaño de fuente para que se ajuste al texto.
10	Highlighted	El color se aplica al texto en la etiqueta cuando se marca el atributo resaltado.
11	Shadow	El valor predeterminado de este atributo es transparente, lo que significa que no aparece ninguna sombra debajo del texto. Sin embargo, podemos mencionar el color de la sombra que se muestra debajo del texto de la etiqueta. Se puede acceder a este valor en tiempo de ejecución utilizando la propiedad shadowColor.
12	ShadowOffset	Esta propiedad mantiene el desplazamiento de sombra. Se puede acceder a este valor utilizando la propiedad shadowOffset en tiempo de ejecución.

Ejemplo

En este ejemplo, agregaremos una etiqueta al ViewController usando el generador de interfaces y personalizaremos la apariencia de la etiqueta en tiempo de ejecución creando la salida de la etiqueta en la clase ViewController.

Para crear una salida, elija el control en la interfaz de usuario para la cual se creará la salida, mantenga presionada la tecla de control y arrastre desde el control a la clase asociada como se muestra en la siguiente imagen.



En la clase ViewController, estableceremos algunas propiedades para que la etiqueta personalice su apariencia en tiempo de ejecución.

ViewController.swift

Salida

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    @IBOutlet weak var textLbl: UILabel!
6.
7.    override func viewDidLoad() {
8.        super.viewDidLoad()
9.        // Do any additional setup after loading the view.
10.        textLbl.text = "Hello World"
11.        textLbl.font = .italicSystemFont(ofSize: 30)
12.        textLbl.backgroundColor = UIColor.blue
13.        textLbl.textAlignment = .center
14.        textLbl.textColor = UIColor.white
15.        textLbl.shadowColor = UIColor.black
16.        textLbl.isHighlighted = true
17.    }
18.}
```



Ejemplo 2

Hacer que la etiqueta se pueda tocar

El siguiente ejemplo hará que la etiqueta creada, en el ejemplo 1, se pueda tocar. Para este propósito, crearemos un objeto de la clase UITapGestureRecognizer.

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    @IBOutlet weak var textLbl: UILabel!
6.    var didTap = true
7.
8.    override func viewDidLoad() {
9.        super.viewDidLoad()
10.        // Do any additional setup after loading the view.
11.        textLbl.text = "Hello World"
12.        textLbl.font = .italicSystemFont(ofSize: 30)
13.        textLbl.backgroundColor = UIColor.blue
14.        textLbl.textAlignment = .center
15.        textLbl.textColor = UIColor.white
16.        textLbl.shadowColor = UIColor.black
17.        textLbl.isHighlighted = true
18.        let tap = UITapGestureRecognizer(target: self, action: #selector(didTextLabelTap(sender:)))
19.        textLbl.isUserInteractionEnabled = true
20.        textLbl.addGestureRecognizer(tap)
21.    }
22.
23.    @objc func didTextLabelTap(sender: UITapGestureRecognizer){
24.        if(didTap){
25.            textLbl.backgroundColor = UIColor.brown
26.            didTap = false
27.        }
28.        else{
29.            textLbl.backgroundColor = UIColor.blue
30.            didTap = true
31.        }
32.    }
33. }
34.}
```

Salida:



Propiedades UILabel

El objeto UILabel contiene las siguientes propiedades, que se pueden usar para personalizar el comportamiento en tiempo de ejecución de la etiqueta.

SN	Propiedad	Tipo	Descripción
1	text	String	Representa el texto de la etiqueta actual.
2	attributedText	NSAttributedString	Representa el texto de la etiqueta con el estilo actual.
3	font	UIFont	Representa la fuente del color.
4 4	TextColor	UIColor	Es el color del texto de la etiqueta.
5 5	textAlignment	NSTextAlignment	Es la alineación del texto dentro del marco de la etiqueta.
6 6	lineBreakMode	NSLineBreakMode	Es la técnica utilizada para ajustar y truncar el texto multilínea.
7 7	isEnabled	Bool	El estado habilitado para usar al dibujar el texto de la etiqueta.
8	adjustFontSizeToFitWidth	Bool	Es un valor booleano que, cuando se establece, reduce el tamaño de la fuente de la etiqueta para que quepa en la cadena del título.
9 9	allowDefaultTighteningForTruncation	Bool	Es un valor booleano que indica si la etiqueta aprieta el texto antes de truncar.
10	baseLineAdjustment	UIBaseLineAdjustment	Controla la forma en que se ajustan las líneas base cuando el texto se contrae para ajustarse al marco de la etiqueta.
11	minimumScaleFactor	CGFloat	Es el factor de escala mínimo para el texto de la etiqueta.
12	numberOfLines	En t	Representa el número máximo de líneas que puede tener el texto de la etiqueta. Ajústelo a 0 para que esto sea ilimitado.
13	highlightedTextColor	UIColor	Representa el color del texto resaltado aplicado a la etiqueta.
14	isHighlighted	Bool	Representa un valor booleano que indica si el texto de la etiqueta está resaltado o no.
15	shadowColor	UIColor	Representa el color de la sombra del texto de la etiqueta.
dieciséis	shadowOffset	CGSize	Representa el desplazamiento de sombra del texto de la etiqueta.
17	preferredMaxLayoutWidth	CGFloat	Es el ancho máximo preferido para una etiqueta multilínea.
18 años	isUserInteractionEnabled	Bool	Es una propiedad de tipo booleano. Cuando se establece en verdadero, permite al usuario interactuar con el texto de la etiqueta.

Button

Es un control que permite al usuario interactuar con la aplicación. Se utiliza para desencadenar los eventos realizados por el usuario. Ejecuta el código personalizado en respuesta a las interacciones del usuario.

```
class UIButton : UIControl
```

Los botones son una de las partes más importantes de las aplicaciones de iOS. Los botones están asociados con las acciones que se realizan cuando el usuario interactúa con el botón. Podemos agregar el botón a la aplicación iOS mediante programación o mediante el generador de interfaces.

Los siguientes pasos se realizan cuando se agrega el botón a la aplicación.

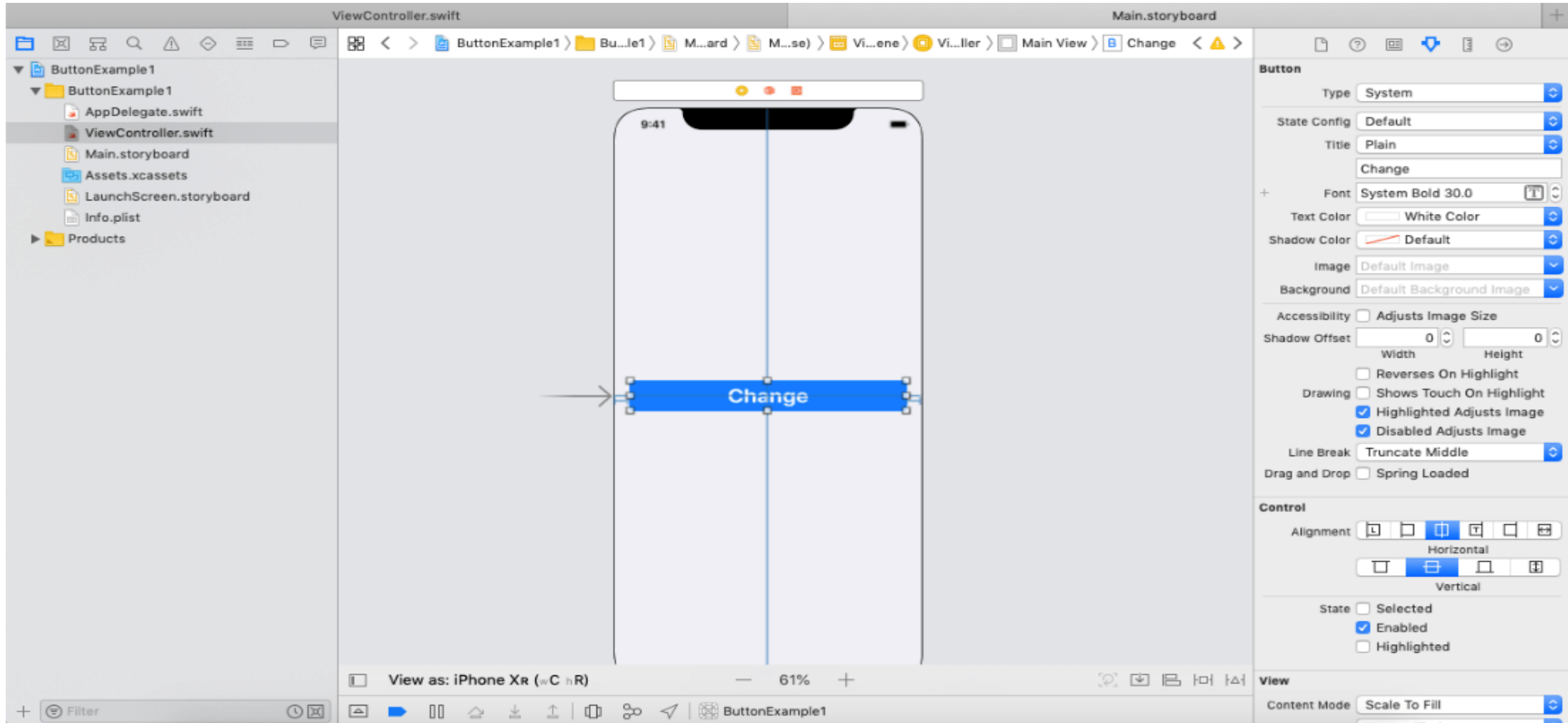
1. Busque el botón en la biblioteca de objetos y arrastre el resultado al guión gráfico.
2. Establezca el tipo de botón en el momento de la creación.
3. Establezca la cadena de título o la imagen para el botón.
4. Defina el tamaño del botón de acuerdo con el contenido del botón.
5. Configure las restricciones para que el botón rija el tamaño y la posición del botón en dispositivos de diferentes tamaños.

Ejemplo

Creemos un ejemplo muy simple en el que agregaremos un botón a nuestro proyecto y crearemos su método de acción en el archivo de clase View Controller para que pueda realizar algunas tareas en los eventos táctiles.

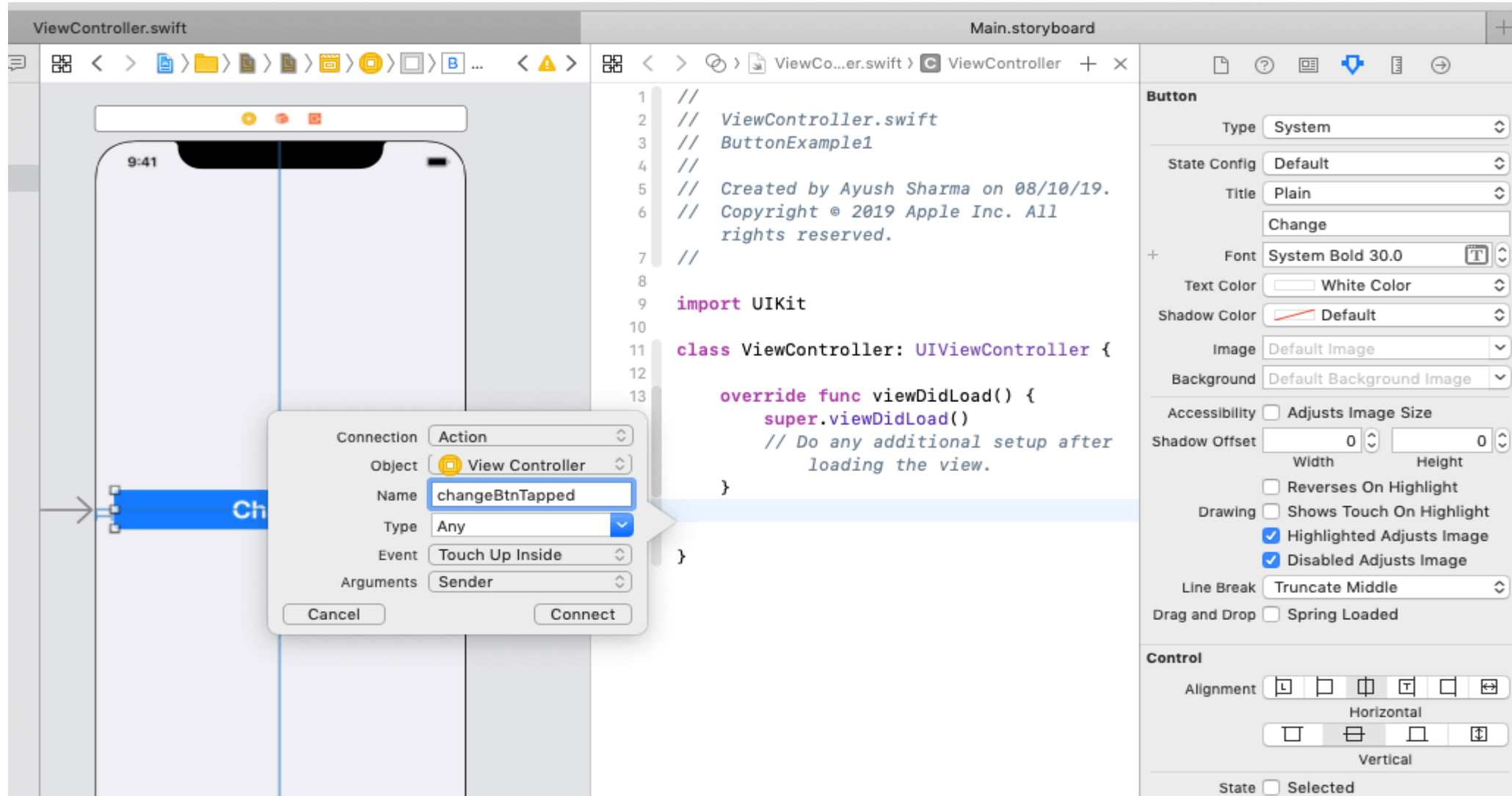
Main.storyboard

En este ejemplo, agregaremos un botón al guión gráfico y le daremos el color de fondo, el tamaño de fuente y la etiqueta del título usando los atributos en el inspector de atributos.



Conexión del action

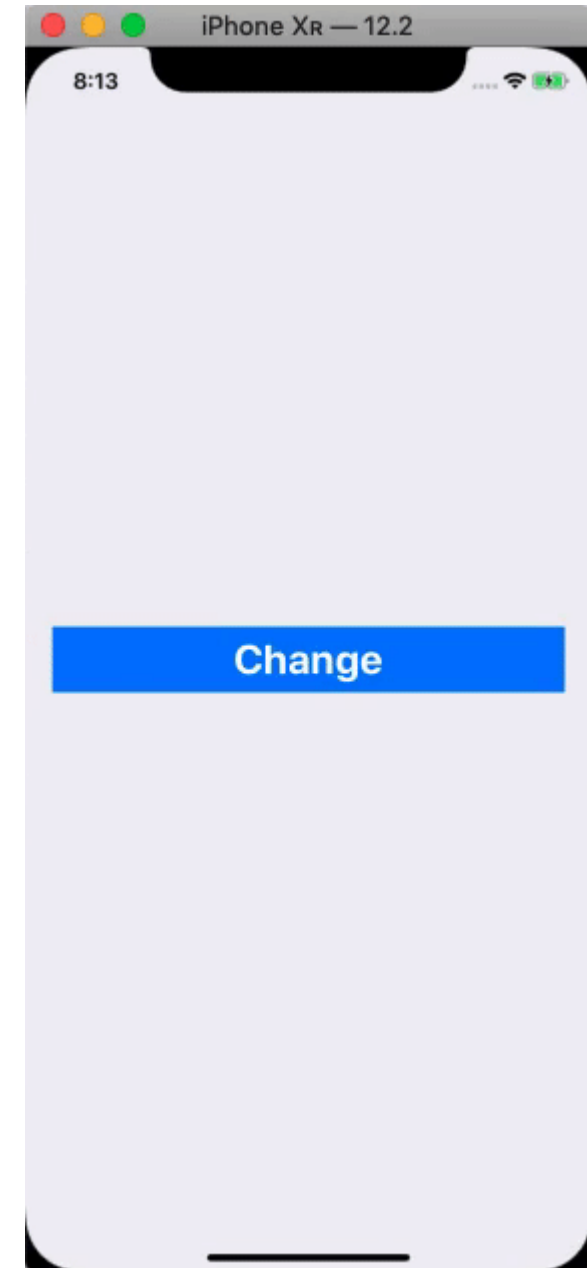
Para realizar cualquier acción al tocar el botón, conectaremos una acción del objeto del botón en el archivo de clase ViewController. En este proyecto, el color de fondo de la vista principal cambiará al tocar el botón dentro del evento.



ViewController.class

```
1.import UIKit
2.class ViewController: UIViewController {
3.    @IBOutlet var mainView: UIView!
4.    var didTap = true
5.
6.    override func viewDidLoad() {
7.        super.viewDidLoad()
8.        // Do any additional setup after loading the view.
9.    }
10.
11.    @IBAction func changeBtnTapped(_ sender: Any) {
12.        if(didTap)
13.        {
14.            mainView.backgroundColor = .orange
15.            didTap = false
16.        }
17.        else{
18.            mainView.backgroundColor = .groupTableViewBackground
19.            didTap = true
20.        }
21.    }
22.}
```

Salida:



Configurar la apariencia del botón

El tipo de botón define su apariencia y comportamiento. Podemos definir el tipo de botón en el archivo del storyboard o utilizando el método `init (type:)`. Los botones pueden ser de dos tipos: Sistema y Personalizado.

Estados del botón

Los botones pueden tener cinco estados

- Default**

Cuando el botón se agrega al `UIView` inicialmente, permanece en el estado predeterminado hasta que el usuario interactúa con él. El estado cambia a otros valores a medida que el usuario interactúa con el botón.

- Highlighted**

Cuando el usuario toca un botón, se mueve al estado resaltado.

- focused**

El botón entra en el estado enfocado cuando recibe el foco del usuario. Podemos cambiar la apariencia del botón en el estado enfocado para que aparezca de manera diferente al estado seleccionado o enfocado.

- Selected**

Este estado no afecta el comportamiento o la apariencia del botón. Sin embargo, este estado se usa para que otros controles como la clase `UISegmentedControl` usen este estado para cambiar su apariencia. Podemos obtener y establecer este valor usando la propiedad `isSelected`.

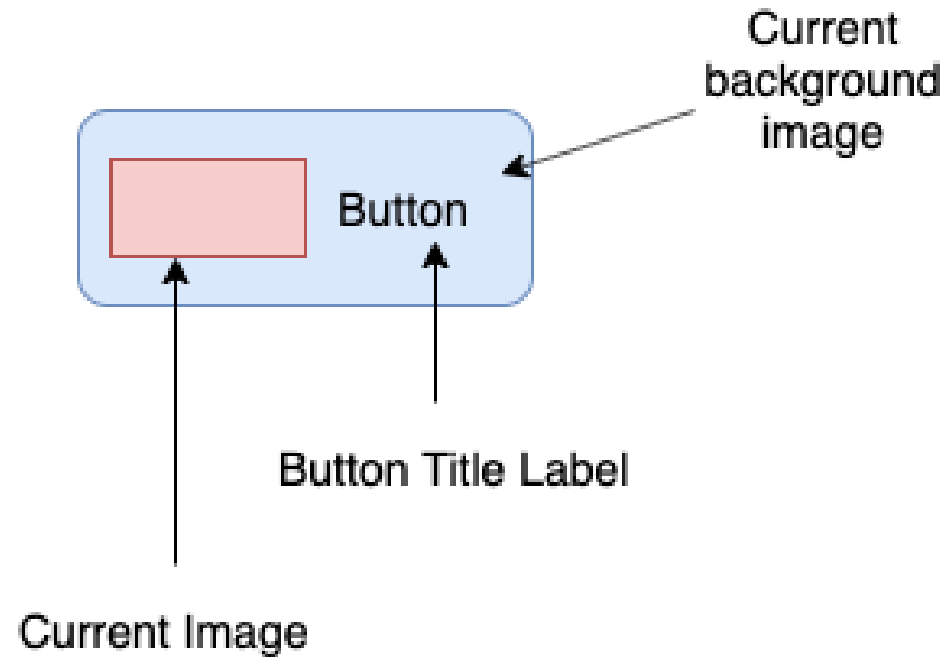
- Disabled**

Es posible que necesitemos desactivar el botón cuando no deseamos que el usuario interactúe con el botón. Este estado se puede establecer y obtener mediante la propiedad `isEnabled`.

Contenido

El contenido del botón indica el comportamiento del botón al usuario. En las aplicaciones de iOS, un botón puede contener una imagen de fondo o un texto de etiqueta de título para especificar el contenido del botón. Es posible que necesitemos configurar los objetos UILabel y UIImageView para administrar el contenido del botón.

Podemos acceder al contenido del botón utilizando las propiedades titleLabel o imageView en el objeto del botón.



Atributos del generador de interfaz

SN	Atributo	Descripción
1	Type	Representa el tipo de botón, que no se puede cambiar en tiempo de ejecución. Solo se puede configurar en el momento de la creación del botón. Se accede mediante la propiedad buttonType.
2	State config	Es el selector de estado del botón. Define el estado del botón para que los cambios se puedan aplicar solo a ese estado.
3	Title	Es el título del botón, que puede ser una cadena simple o una cadena atribuida.
4 4	Tint, Font y attribute	Estos atributos se aplican a la cadena del título del botón, como color de tinte, fuente, color de texto, color de sombra, etc.
5 5	Image	Es la imagen de primer plano del botón.
6 6	Background	Es la imagen de fondo del botón. Se muestra detrás del título y la imagen en primer plano.

Atributos de apariencia

SN	Atributo	Descripción
1	Shadow Offset	Es el desplazamiento de sombra aplicado a las cadenas de título del botón. Este atributo se puede establecer en tiempo de ejecución mediante el uso de la propiedad shadowOffset en el título Etiqueta del botón.
2	Shadow Offset	Representa el comportamiento de dibujo del botón. En el creador de interfaces, podemos establecer tres opciones, es decir, showTouchWhenHighlighted, ajustarImageWhenHighlighted y ajustarImageWhenDisabled.
3	Line Break	Es el modo de salto de línea de la etiqueta del título del botón.

Atributos de inserción de borde

SN	Atributo	Descripción
1	Edge	Son las inserciones de borde para configurar. Podemos configurar las inserciones de borde separadas para el contenido general del botón.
2	Inset	Representa los valores insertados. Se puede acceder a estos valores mediante la propiedad <code>contentEdgeInsets</code> , <code>titleEdgeInsets</code> e <code>imageEdgeInsets</code> .

TextField

Se puede definir como un objeto que se utiliza para mostrar un área de texto editable en la interfaz. Los campos de texto se utilizan para obtener la entrada de texto del usuario.

clase UITextField: UIControl

Como hemos discutido para la etiqueta, también podemos personalizar el campo de texto dando atributos. Hay escenarios en los que también podríamos necesitar configurar el teclado, por ejemplo, para restringir que el usuario ingrese solo los números. Textfield utiliza el mecanismo de acción de destino y un objeto delegado para informar los cambios realizados durante la edición. Los siguientes pasos se utilizan para configurar el campo de texto.

1. Busque el TextField en la biblioteca de objetos y arrastre el resultado al guión gráfico.
2. Configure uno o más objetivos y acciones para el campo de texto.
3. Personalice los atributos relacionados con el teclado para el campo de texto.
4. Realice tareas de objeto de delegado de campo de texto para informar los cambios realizados durante la edición.
 1. El método de delegado de campo de texto se llama cuando el usuario cambia el contenido del campo de texto.
 2. Validar el texto ingresado por el usuario.
 3. Se llama al método delegado cuando se toca el botón de retorno del teclado.
5. Cree una salida del campo de texto en la clase ViewController correspondiente.

Ejemplo

Aquí, crearemos un formulario utilizando campos de texto y etiquetas. También aplicaremos algunas validaciones en el campo de texto. Si la validación falla, se mostrará una alerta al usuario; de lo contrario, se mostrará un mensaje de éxito al usuario.

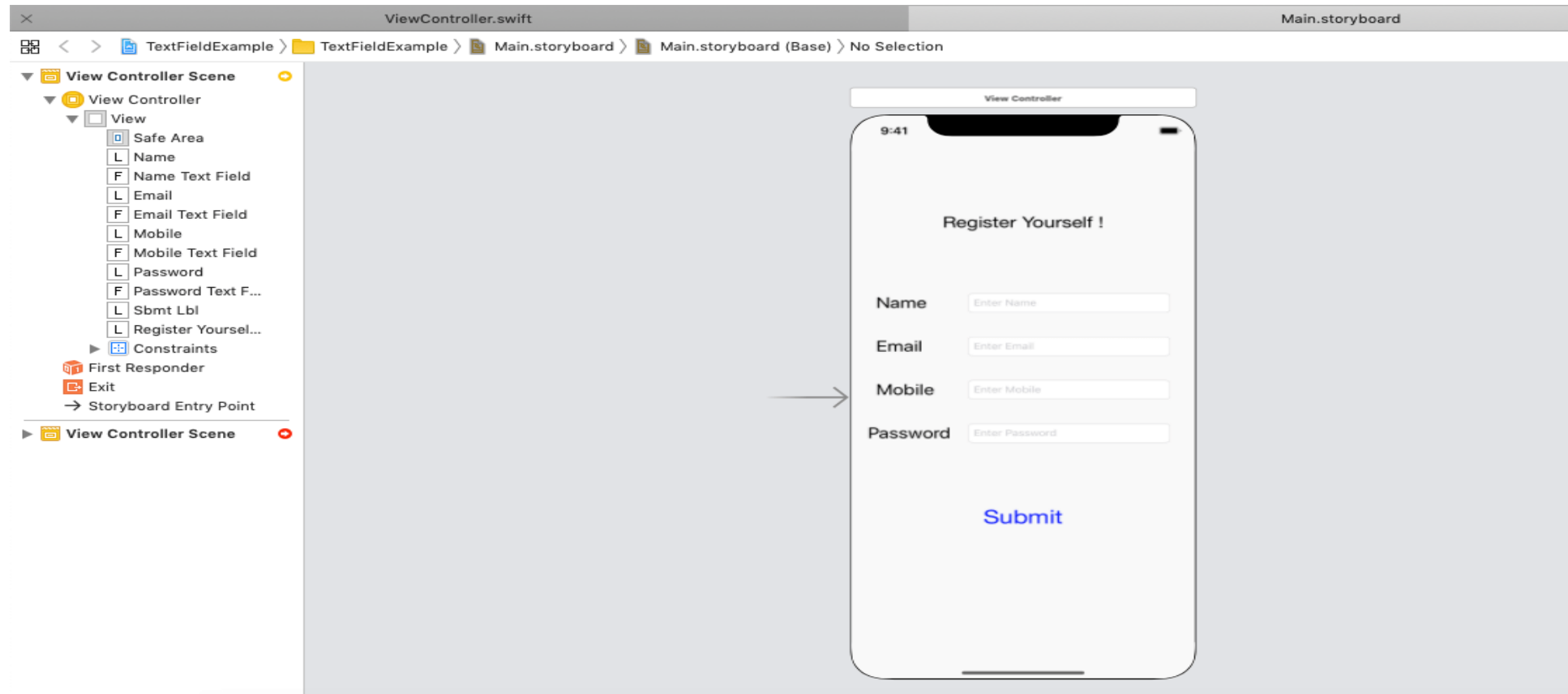
Constructor de interfaz

Para crear la interfaz para el proyecto, utilizaremos `textField` para obtener los datos del usuario, asociados con las etiquetas para mostrar los mensajes al usuario.

No utilizaremos `Button` en la aplicación, ya que aún no está cubierto en este tutorial; en su lugar, estamos usando etiquetas en las que se puede hacer clic para activar los eventos.

Cree la interfaz buscando los objetos apropiados en la biblioteca de objetos y arrastre el resultado al storyboard.

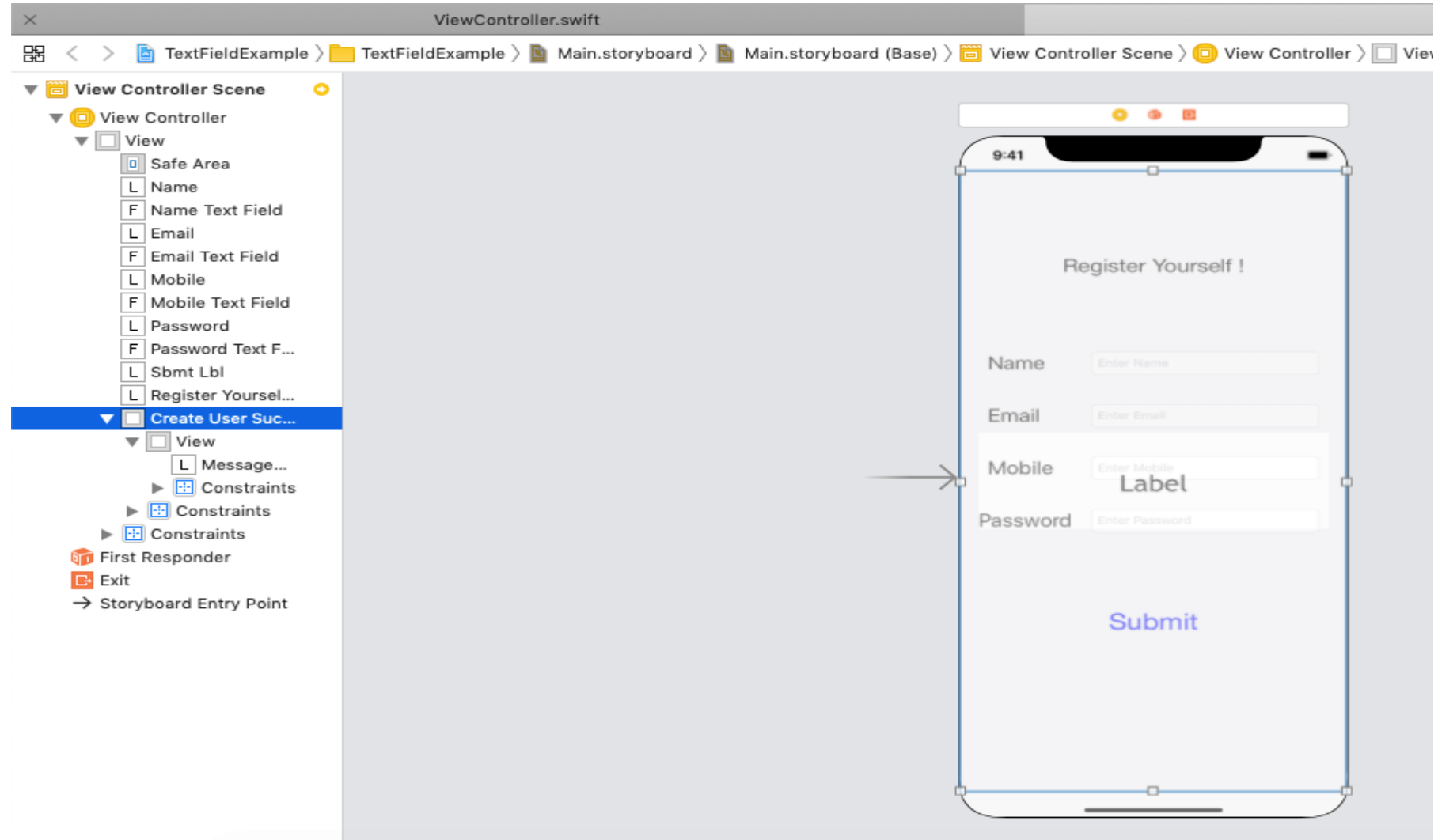
La siguiente Vista se crea en el proyecto para obtener la información del usuario.



¿Cómo mostrar el mensaje de éxito en el registro cumplimentado?

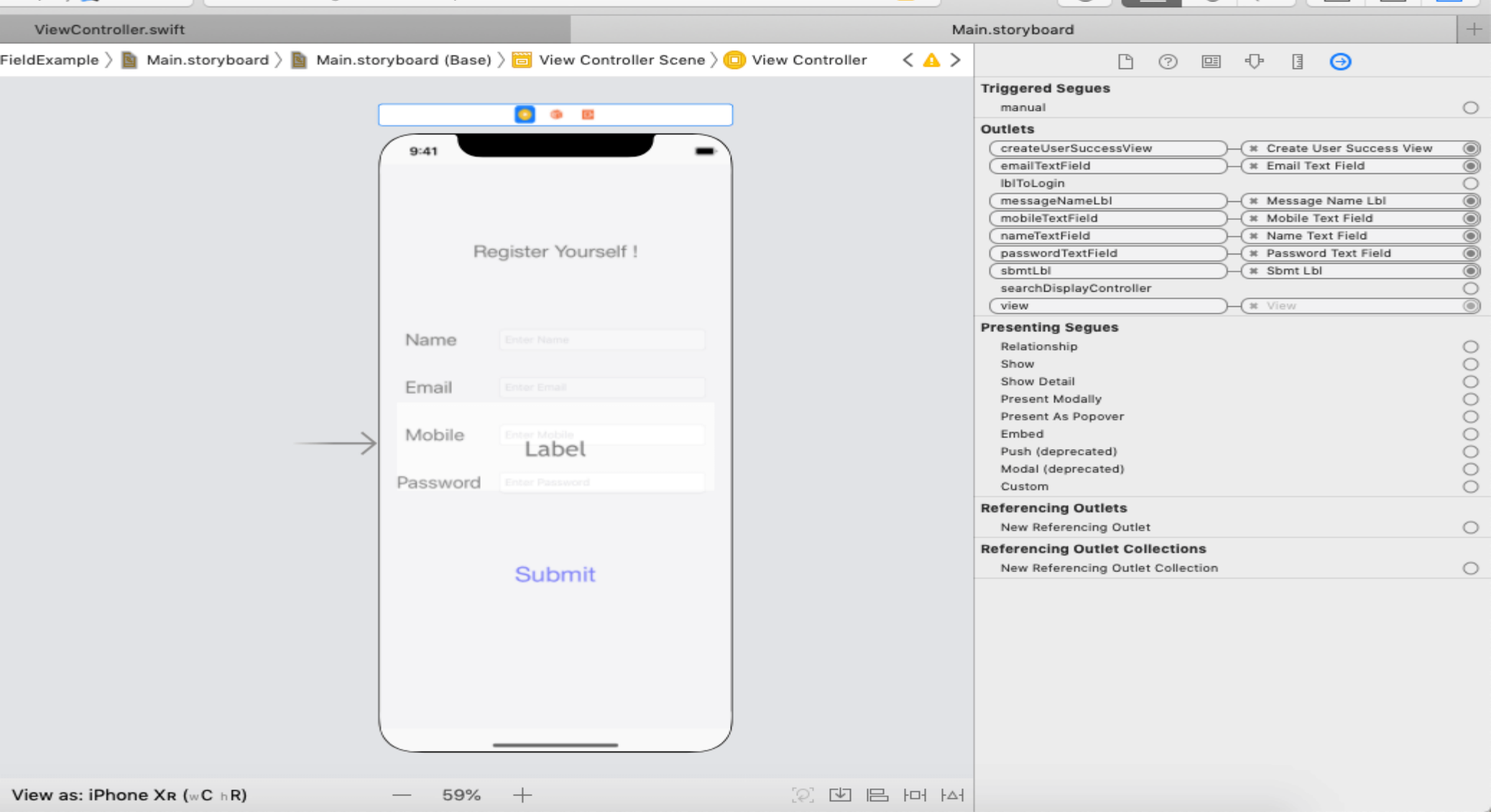
En el desarrollo de iOS, generalmente creamos otra pantalla al completar con éxito cualquier tarea. Sin embargo, en este ejemplo, crearemos una UIView que contenga el mensaje de éxito en la misma pantalla y lo mantendremos oculto hasta que la validación tenga éxito en la etiqueta de envío tocada.

La interfaz después de agregar createUserSuccessView se muestra en la siguiente imagen.



Outlets

Las salidas se crean de los objetos en la clase Associated View Controller. Todas las salidas de conexión se pueden mostrar en el inspector de conexiones en XCode, como se muestra en la siguiente imagen.



ViewController.swift

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.    @IBOutlet weak var createUserSuccessView: UIView!
5.
6.    @IBOutlet weak var nameTextField: UITextField!
7.
8.    @IBOutlet weak var emailTextField: UITextField!
9.
10.    @IBOutlet weak var passwordTextField: UITextField!
11.
12.    @IBOutlet weak var mobileTextField: UITextField!
13.
14.    @IBOutlet weak var sbmtLbl: UILabel!
15.
16.    @IBOutlet weak var messageNameLbl: UILabel!
17.
18.    override func viewDidLoad() {
19.        super.viewDidLoad()
20.        // Do any additional setup after loading the view.
21.        setInitViews()
22.        sbmtLbl.isUserInteractionEnabled = true
23.        let tap = UITapGestureRecognizer(target: self, action: #selector(sbmtBtnTapped(sender:)))
24.        sbmtLbl.addGestureRecognizer(tap)
25.    }
26.    func setInitViews(){
27.        nameTextField.becomeFirstResponder()
28.        emailTextField.delegate = self
29.        mobileTextField.delegate = self
30.        nameTextField.delegate = self
31.        passwordTextField.delegate = self
32.    }
33. }
34.
35.
36. @objc func sbmtBtnTapped(sender: UITapGestureRecognizer){
37.     if(nameTextField.text?.isEmpty ?? false || emailTextField.text?.isEmpty ?? false || passwordTextField.text?.isEmpty ?? false || passwordTextField.text?.isEmpty ?? false){
38.         let alert = UIAlertController(title: nil, message: "Please fill all the details", preferredStyle: .alert)
39.         let action = UIAlertAction(title: "OK", style: .default) { (action) in
40.             self.dismiss(animated: true, completion: nil)
41.         }
42.         alert.addAction(action)
43.         self.present(alert, animated: true, completion: nil)
44.     }
45.     else{
46.         messageNameLbl.text = "Hi " + (nameTextField.text ?? "")
47.         createUserSuccessView.isHidden = false
48.     }
49. }
50. }
51.}
52.
53.
54.extension ViewController:UITextFieldDelegate{
55.
56.    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString string: String) -> Bool {
57.        if(textField == mobileTextField){
58.            let currrentCharacterCount = textField.text?.count ?? 0
59.            if range.length + range.location > currrentCharacterCount {
60.                return false
61.            }
62.            let newLength = currrentCharacterCount + string.count - range.length
63.            return newLength <= 10
64.        }
65.        else{
66.            return true
67.        }
68.    }
69.}
```

Salida:

iPhone XR — 12.2

12:59

Register Yourself !

Name

Email

Mobile

Password

Submit

Configurar el teclado

El campo de texto puede convertirse en el primer respondedor llamando al método BecomeFirstResponder () en la salida del campo de texto. Cuando el campo de texto se convierte en el primer respondedor, el teclado se vuelve automáticamente visible y la entrada del teclado está vinculada al campo de texto. Cuando el usuario toca el campo de texto, el campo de texto se convierte en el primer respondedor, es decir, el teclado se vuelve visible.

Para descartar el teclado, se llama al método resignFirstResponder () en la salida del campo de texto. Es posible que debamos descartar el teclado en respuesta a algunas interacciones específicas. El teclado se descarta automáticamente cuando el usuario presiona la tecla de retorno.

El estado de edición del campo de texto se ve afectado por la apariencia y el descarte del teclado. Cuando el usuario comienza a editar el campo de texto en la apariencia del teclado, el campo de texto envía la notificación correspondiente a su delegado. Los métodos de delegado también se notifican cuando comienza y finaliza la edición.

Métodos de delegado de TextField

SN	Firma del método	Descripción
1	func textFieldShouldBeginEditing (UITextField) -> Bool	Le pregunta al delegado si la edición debe comenzar el campo de texto respectivo.
2	func textFieldDidBeginEditing (UITextField)	Le dice al delegado que la edición se inicia en el campo de texto.
3	func textFieldShouldEndEditing (UITextField) -> Bool	Le pide al delegado que finalice la edición en el campo de texto.
4 4	func textFieldDidEndEditing (UITextField, motivo: UITextField.DidEndEditingReason)	Le dice al delegado que la edición se detuvo para el campo de texto especificado.
5 5	func textFieldDidEndEditing (UITextField)	Son los métodos sobrecargados los que también hacen lo mismo que el anterior.
6 6	func textField (UITextField, shouldChangeCharactersIn: NSRange, replaceString: String) -> Bool	Le pregunta al delegado si el contenido actual del campo de texto debe cambiarse.
7 7	func textFieldShouldClear (UITextField) -> Bool	Le pregunta al delegado si se debe eliminar el contenido actual del campo de texto.

Atributos del generador de interfaz

Atributos de TextField

SN	Atributo	Descripción	Propiedad
1	text	Representa el texto actual que contiene el archivo de texto. Se puede acceder mediante la propiedad de texto del campo de texto en tiempo de ejecución.	text
2	color	Representa el color del texto del campo de texto. Se puede acceder a esto utilizando la propiedad textColor de TextField.	textColor
3	font	Representa la fuente del texto del campo de texto. Se puede acceder a esto utilizando la propiedad de fuente del objeto de campo de texto.	font
4 4	alignment	Representa la alineación del texto dentro del área de edición.	textAlignment
5 5	placeholder	Representa la cadena de marcador de posición del campo de texto. Cuando el campo de texto está vacío, esta cadena se muestra para guiar al usuario.	placeholder
6 6	background	Representa la imagen de fondo para mostrar cuando el campo de texto está habilitado. La imagen de fondo se muestra detrás del contenido del campo de texto.	background
7 7	disabled	Representa la imagen de fondo para mostrar cuando el campo de texto está desactivado.	disabledBackground
8	Border style	Representa el estilo visual del borde del campo de texto.	borderStyle
9 9	Clear button	El botón Borrar es una vista superpuesta que el usuario puede tocar para eliminar todo el contenido del campo de texto. Este atributo se puede usar para establecer la apariencia del botón Borrar dentro del campo de texto.	clearButtonMode
10	Min Font size	El tamaño de fuente mínimo para el texto del campo de texto. Cuando la opción Ajustar para ajustar está habilitada, el campo de texto varía automáticamente el tamaño de fuente para garantizar la máxima legibilidad del texto.	minFontSize

Atributos del teclado

SN	Atributo	Descripción	Propiedad
1	Capitalization	Este atributo aplica el estilo de mayúsculas automático al texto escrito dentro del campo de texto.	autocapitalizationType
2	Correction	Este atributo se usa para determinar si la corrección automática está habilitada o no mientras se escribe.	autocorrectionType
3	Spell Checking	Se utiliza para determinar si la corrección ortográfica se habilitó o no mientras se escribe.	spellCheckingType
4 4	Keyboard type	Esta propiedad establece el tipo de teclado que se muestra al escribir.	keyboardType
5 5	Appearance	Se utiliza para establecer la apariencia del campo de texto. Especifica un teclado oscuro o claro.	keyboardAppearance
6 6	Run key	Se utiliza para establecer el tipo de la tecla de retorno para mostrar en el teclado.	returnKeyType

Selector de fechas

DatePicker es un control utilizado en aplicaciones iOS para obtener los valores de fecha y hora del usuario. Podemos permitir que el usuario ingrese el tiempo en un punto o intervalo de tiempo.

clase UIDatePicker: UIControl

Utilice los siguientes pasos para agregar el DatePicker a la interfaz.

1. Establezca la UIView, es decir (etiqueta, campo de texto, botón, etc.) en la que desea obtener la fecha y la hora del usuario.
2. Busque DatePicker en la biblioteca de objetos y arrastre el resultado al guión gráfico.
3. Establecer el modo selector de fecha
4. Proporcione opciones de configuración adicionales, como fechas mínimas y máximas, si es necesario.
5. Cree una salida de acción para el selector de fecha.
6. Configure reglas de diseño automático para datePicker para establecer la posición en diferentes dispositivos.

El selector de fecha solo se usa para la selección de fecha y hora de la lista.

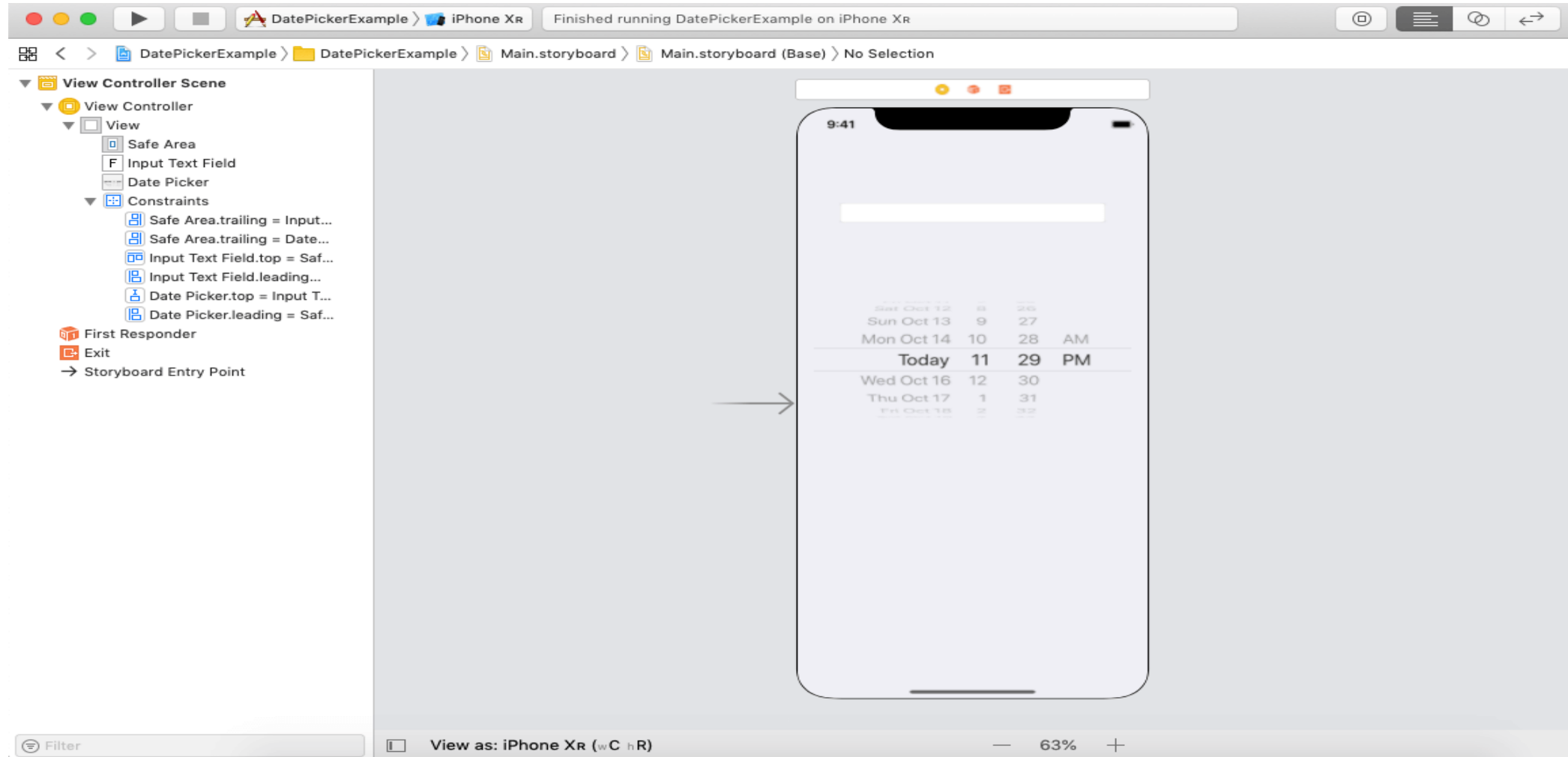
Ejemplo

En este ejemplo, crearemos un campo de texto en el que permitiremos que un usuario ingrese una fecha seleccionando cualquier fecha del selector de fechas.

Para agregar el selector de fecha al gui3n gr3fico, buscaremos el selector de fecha en la biblioteca de objetos y arrastraremos el resultado al gui3n gr3fico.

Interface Builder

La siguiente imagen muestra el generador de interfaces; Creamos en el ejemplo. Hemos creado la salida para el campo de texto y el selector de fecha en el archivo ViewController.swift.

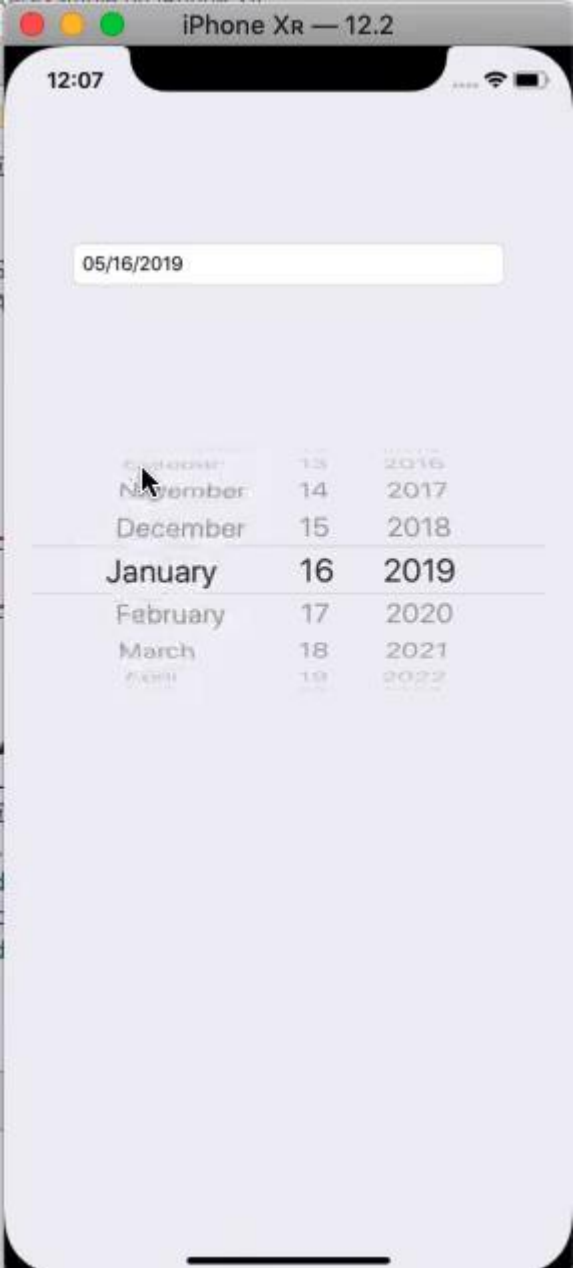


ViewController.swift

En ViewController.swift, acabamos de asignar la propiedad `inputView` del campo de texto de entrada al selector de fecha, y la conexión de acción del selector de fecha se activa cada vez que se cambia el valor del selector de fecha, lo que establece el texto del campo de texto en la fecha del selector de fecha . En este ejemplo, hemos establecido el modo `datepicker` en fecha; sin embargo, podemos establecerlo en `dateAndTime` u hora para obtener los valores apropiados de fecha y hora.

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    @IBOutlet weak var inputTextField: UITextField!
6.
7.    @IBOutlet weak var datePicker: UIDatePicker!
8.
9.    let dateFormatter = DateFormatter()
10.
11.    override func viewDidLoad() {
12.        super.viewDidLoad()
13.        // Do any additional setup after loading the view.
14.        dateFormatter.dateFormat = "MM/dd/yyyy"
15.        inputTextField.inputView = datePicker
16.        datePicker.datePickerMode = .date
17.        inputTextField.text = dateFormatter.string(from: datePicker.date)
18.
19.    }
20.
21.    @IBAction func datePickerValueChanged(_ sender: UIDatePicker) {
22.
23.        inputTextField.text = dateFormatter.string(from: sender.date)
24.        view.endEditing(true)
25.    }
26.}
```

Salida:



Atributos del generador de interfaz

Atributos centrales

SN	Atributo	Descripción
1	Mode	Representa el modo DatePicker. Se utiliza para determinar si el selector de fecha mostrará la fecha, hora, fecha y hora o un intervalo de cuenta regresiva. Se puede acceder a este en tiempo de ejecución utilizando la propiedad datePickerMode.
2	Locate	Esto representa el entorno local asociado con el selector de fecha. Esta propiedad anula la configuración regional predeterminada del sistema. Se puede acceder a este en tiempo de ejecución utilizando la propiedad local.
3	interval	Representa la granularidad de la ruleta del minuto. El valor predeterminado es 1 y el valor máximo es 30. Este valor debe ser un divisor de 60. Se puede acceder en tiempo de ejecución utilizando la propiedad minuteInterval.

Atributos de fecha

SN	Atributo	Descripción
1	Date	Representa la fecha en que el selector de fecha se mostrará inicialmente. Podemos establecer esta propiedad en tiempo de ejecución.
2	Constraints	Representa el rango de las fechas que se pueden seleccionar. Podemos configurar la propiedad <code>minimalDate</code> y <code>maximumDate</code> para configurar el rango.
3	Timer	Es el valor inicial del selector de fecha cuando se muestra en modo de temporizador de cuenta regresiva.

Control deslizable

Un control deslizable se puede definir como un UIControl, que proporciona el rango contiguo de valores en una sola escala para el usuario, del cual se le pide al usuario que seleccione un valor único. Se supone que el usuario debe mover el pulgar sobre el control deslizable. El control deslizable está conectado con el método de acción, que se notifica cada vez que el usuario mueve el pulgar hacia un control deslizable. El valor del control deslizable se puede recuperar cada vez que se llama al método de acción.

El control deslizable se declara de la siguiente manera.

clase UISlider: UIControl

Los siguientes pasos se utilizan para agregar los controles deslizantes al generador de interfaces

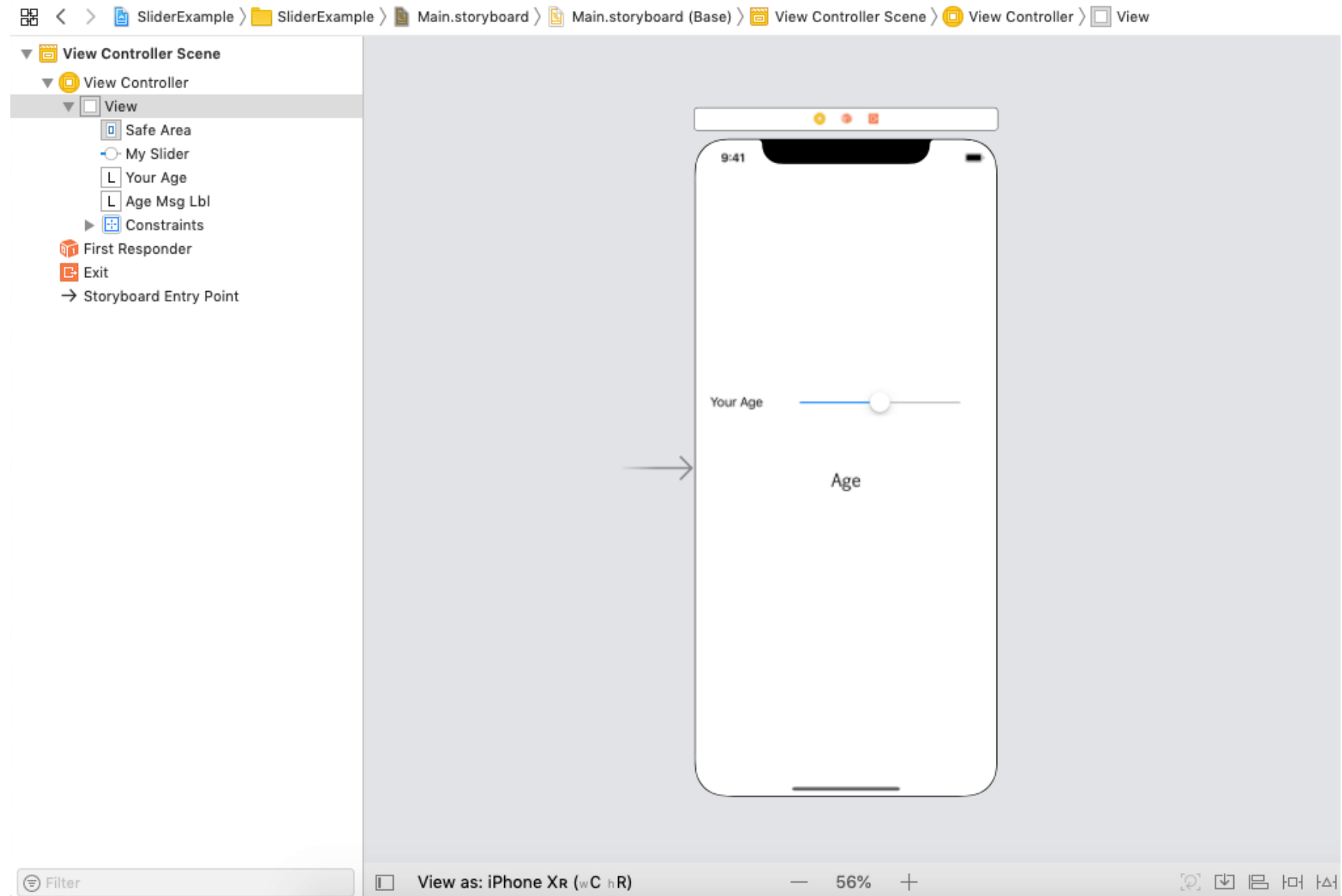
1. Busque el control deslizable en la biblioteca de objetos y arrastre el resultado al guión gráfico.
2. Dé los atributos al control deslizable como el rango de los valores representados por el control deslizable, el color del tinte, las imágenes de límite, etc. desde el guión gráfico o mediante programación.
3. Defina la lógica que se ejecutará cuando el valor del control deslizable se cambie cada vez en el método de acción del control deslizable.
4. Configure reglas de diseño automático para controlar el tamaño y la posición de los controles deslizantes en diferentes tamaños de pantalla.

Ejemplo

En este ejemplo, proporcionaremos el control deslizante al usuario en el que un usuario puede seleccionar entre diferentes valores para elegir su edad.

Interface Builder

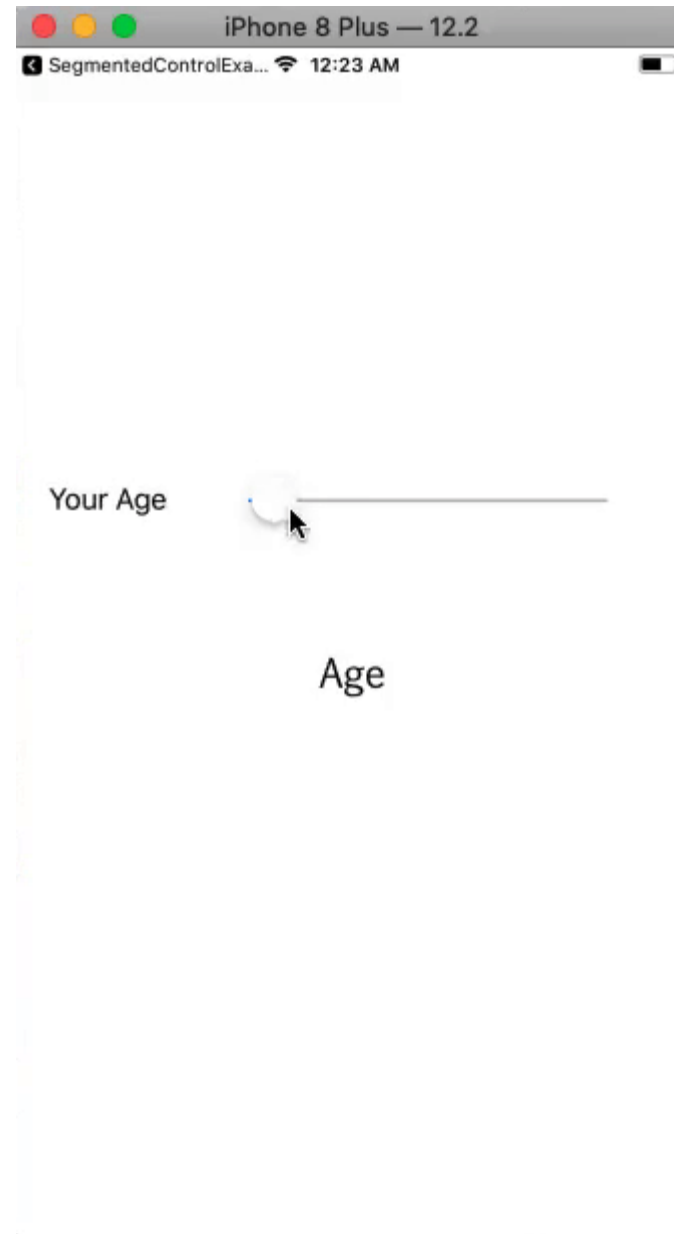
En este ejemplo, hemos creado un gui3n gr3fico muy simple, en el que hemos utilizado el control deslizante y la etiqueta para mostrar el valor del valor actual del control deslizante. La siguiente imagen muestra el generador de interfaz (gui3n gr3fico) creado en el proyecto. El control deslizante est3 conectado al m3todo de acci3n, que se utiliza para establecer el texto de la etiqueta del mensaje de acuerdo con el valor actual del control deslizante.



ViewController.swift

```
1.import UIKit
2.class ViewController: UIViewController {
3.    @IBOutlet weak var mySlider: UISlider!
4.    @IBOutlet weak var ageMsgLbl: UILabel!
5.    override func viewDidLoad() {
6.        super.viewDidLoad()
7.        // Do any additional setup after loading the view.
8.        mySlider.minimumValue = 0
9.        mySlider.maximumValue = 60
10.    }
11.
12.    @IBAction func sliderValueChanged(_ sender: UISlider) {
13.        let roundedValue = round(sender.value)
14.        sender.value = roundedValue
15.        ageMsgLbl.text = "Your Age is "+Int(sender.value).description
16.    }
17.
18.}
```

Salida:



Atributos del generador de interfaz

Atributos centrales

SN	Atributo	Descripción
1	Value (minimum/ maximum)	Representa el valor flotante, que se especifica en los extremos del control deslizante. El valor mínimo representa el extremo inicial del control deslizante, mientras que el máximo representa el extremo final del control deslizante.
2	Value (current)	Representa el valor inicial del control deslizante, que cambia cuando el usuario interactúa con el control deslizante. Existe entre los valores mínimo y máximo. Se puede acceder a esto en tiempo de ejecución utilizando la propiedad value en el objeto deslizante.

Atributos de apariencia

SN	Atributo	Descripción
1	Min Image	Representa la imagen especifica el extremo delantero del control deslizante. Se puede acceder a esto utilizando la propiedad minimumValueImage en tiempo de ejecución.
2	Max Image	Representa la imagen especifica el extremo final del control deslizante. Se puede acceder a esto utilizando la propiedad maximumValueImage en tiempo de ejecución.
3	Min Track Tint	Es el color del tono de la pista del lado delantero del control deslizante. Se puede acceder a esto utilizando la propiedad minimumTrackTintColor en tiempo de ejecución.
4 4	Max Track Tint	Es el color del tono de la pista del lado posterior del control deslizante. Se puede acceder a esto utilizando la propiedad maximumTrackTintColor en tiempo de ejecución.
5 5	Thumb Tint	Es el color del tinte del pulgar del control deslizante. Se puede acceder a esto utilizando la propiedad thumbTintColor en tiempo de ejecución.

Stepper

Es un tipo de UIControl que se usa para aumentar y disminuir el valor. El paso a paso consta de dos botones. Se asocia con un valor que aumenta o disminuye repetidamente al mantener presionado dos de los botones una vez a la vez. La velocidad del cambio depende de la duración en que el usuario presiona el control.

clase UIStepper: UIControl

Podemos establecer los valores máximos y mínimos para el UIStepper; sin embargo, el valor máximo siempre será mayor que el valor mínimo. Si el valor máximo es menor que el valor mínimo, ambos valores se vuelven iguales.

Pasos para agregar Stepper al guión gráfico

1. Configure una etiqueta, un campo de texto o un botón cuyo texto deba cambiarse al cambiar el valor paso a paso.
2. Busque el UIStepper en la biblioteca de objetos y arrastre el resultado al guión gráfico.
3. Cree la salida y la salida de acción del paso a paso en el archivo ViewController.swift.
4. Configure el paso a paso mediante programación en el archivo ViewController.swift.
5. Establezca las reglas de diseño automático para el UIStepper para gobernar la posición y el tamaño del paso a paso en los diferentes tamaños de pantalla.

Ejemplo

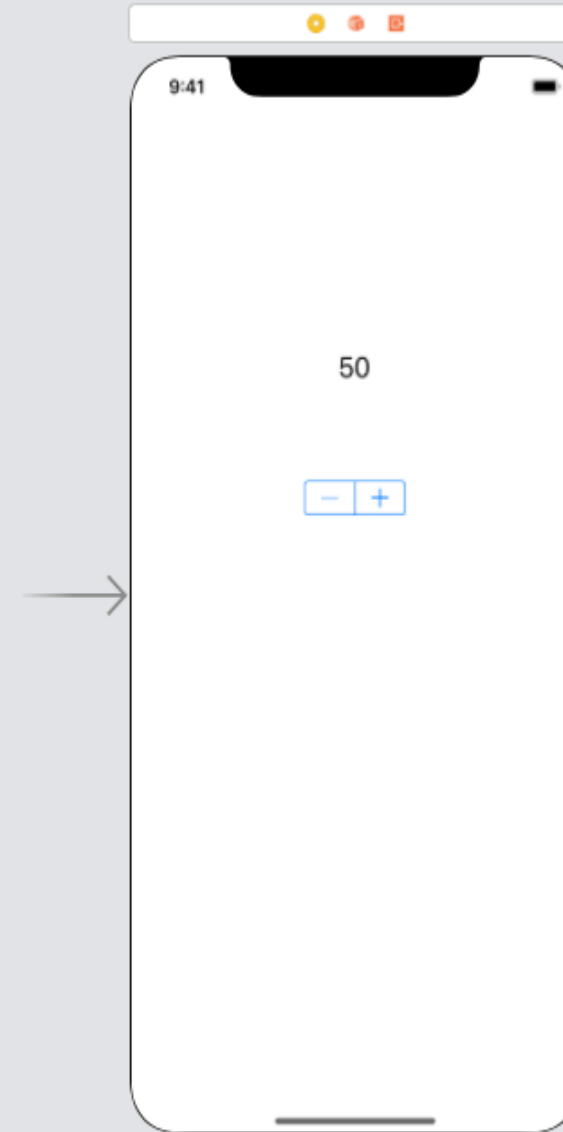
En este ejemplo, estamos usando una etiqueta y el UIStepper. Es un ejemplo simple, en el que el texto de la etiqueta se cambia equivalente al cambio en el valor paso a paso.

Interface Builder

Hemos utilizado la etiqueta y el UIStepper para crear el generador de interfaces que se muestra en la siguiente imagen.

StepperExample > StepperExample > Main.storyboard > Main.storyboard (Base) > No Selection

- View Controller Scene
 - View Controller
 - View
 - Safe Area
 - Value Lbl
 - Stepper
 - Constraints
 - height = 29
 - width = 94
 - Constraints
 - Safe Area.trailing = Valu...
 - Value Lbl.top = Safe Are...
 - Value Lbl.leading = Safe...
 - Stepper.top = Value Lbl...
 - Stepper.centerX = centerX
 - First Responder
 - Exit
 - Storyboard Entry Point



Filter

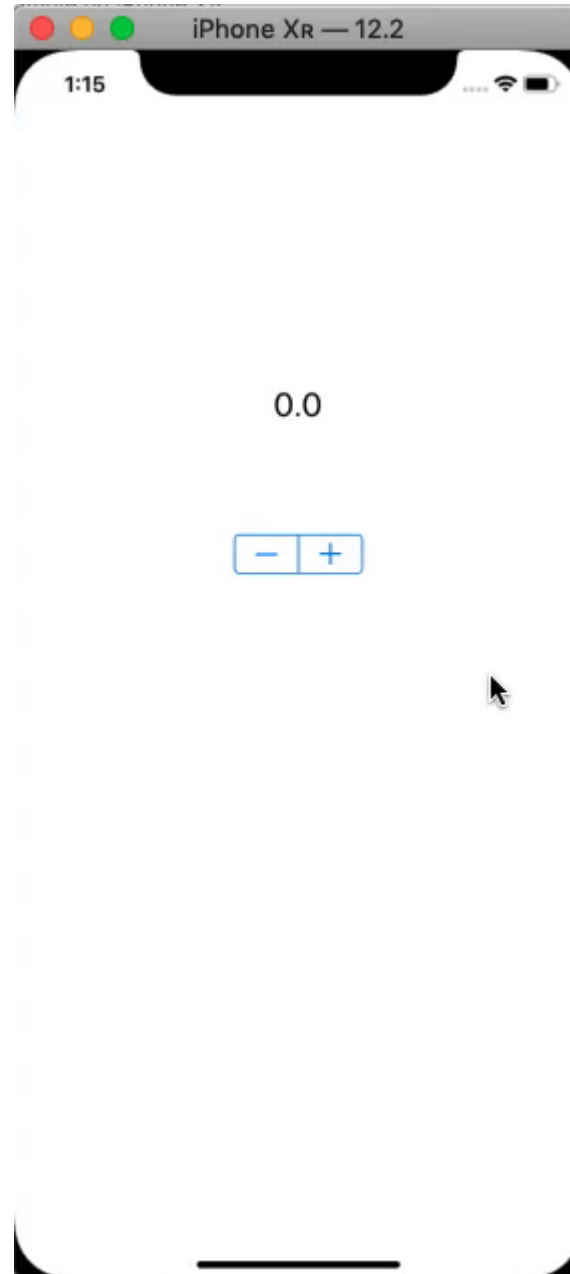
View as: iPhone XR (w C h R)

60% +

ViewController.swift

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    @IBOutlet weak var valueLbl: UILabel!
6.    @IBOutlet weak var stepper: UIStepper!
7.
8.    override func viewDidLoad() {
9.        super.viewDidLoad()
10.        // Do any additional setup after loading the view.
11.        stepper.autorepeat = true
12.        stepper.isContinuous = true
13.        valueLbl.text = stepper.value.description
14.        stepper.maximumValue = 20
15.        stepper.minimumValue = -20
16.    }
17.
18.    @IBAction func stepperValueChanged(_ sender: UIStepper) {
19.        valueLbl.text = sender.value.description
20.    }
21.}
```

Salida:



Switch

Switch se puede definir como la uicontrol, que proporciona opciones binarios para el usuario, ya sea **on** u **off** . El estado de un conmutador se gestiona mediante propiedades y métodos definidos en la clase UISwitch, que es la subclase de UIControl.

La clase UISwitch se declara de la siguiente manera.

clase UISwitch: UIControl

puede tener solo un estado a la vez, ya sea encendido o apagado. Cuando el usuario intenta cambiar el estado del conmutador, se genera el evento `valueChanged` y se envía una llamada de acción a la conexión de acción asociada con el swift.

Podemos personalizar la apariencia del swift usando las propiedades definidas en la clase UISwitch.

Se puede agregar un interruptor a la interfaz mediante los siguientes pasos.

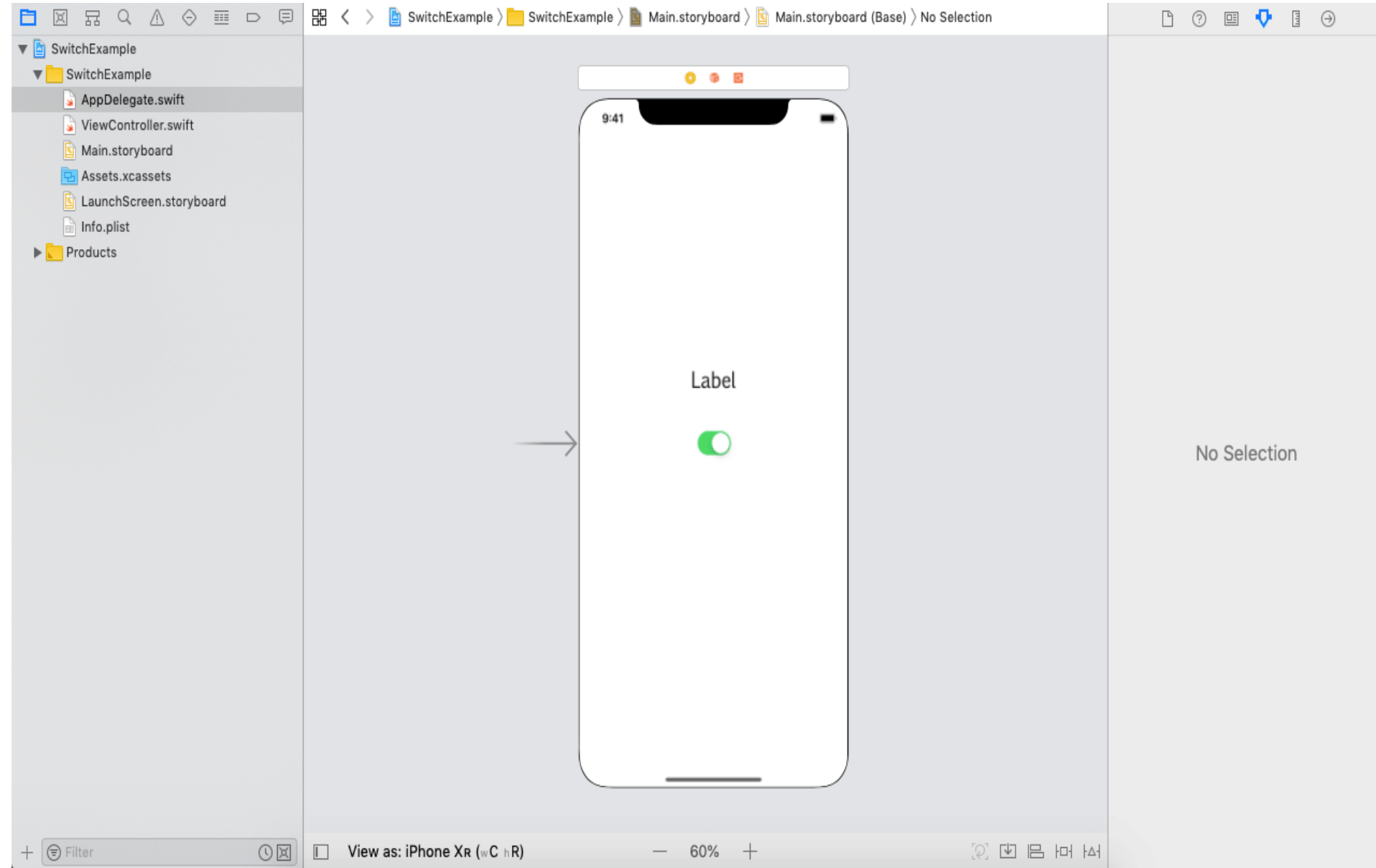
- 1.Cree el objeto de la clase UISwitch o búsquelo en la biblioteca de objetos y arrastre el resultado al generador de guiones gráficos.
- 2.Cree una salida del conmutador en la clase ViewController para personalizar su apariencia en tiempo de ejecución.
- 3.Cree un método de conexión de acción del conmutador en la clase ViewController, que se puede invocar en tiempo de ejecución cuando se desencadena el evento `valueChanged` para el conmutador.

Ejemplo

Aquí, crearemos un ejemplo muy simple, en el que mantendremos el estado del conmutador y crearemos una función que acepte la devolución de llamada cuando se cambie el estado del conmutador.

Interface Builder

En este ejemplo, hemos creado un guión gráfico muy simple en el que hemos utilizado el interruptor y la etiqueta. Aquí, usaremos la etiqueta para mostrar el estado del interruptor, ya sea que esté encendido o apagado. La etiqueta y el interruptor están conectados a las salidas en el archivo ViewController.swift.



ViewController.swift

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    @IBOutlet weak var msgLbl: UILabel!
6.
7.    @IBOutlet weak var mySwitch : UISwitch!
8.
9.    override func viewDidLoad() {
10.        super.viewDidLoad()
11.        // Do any additional setup after loading the view.
12.    }
13.
14.    @IBAction func switchValueChanged(_ sender: UISwitch) {
15.        if(mySwitch.isOn){
16.            msgLbl.text = "Switch is On"
17.        }
18.        else {
19.            msgLbl.text = "Switch is Off"
20.        }
21.    }
22.
23.}
```

Salida:



Segment Control

El control de segmento se puede definir como el control horizontal, que controla múltiples segmentos donde un botón discreto controla cada segmento. Se puede usar un control de segmento para mostrar múltiples vistas dentro de un solo controlador de vista, donde cada vista se puede mostrar usando un botón discreto.

El control de segmento se declara de la siguiente manera.

clase UISegmentedControl: UIControl

El UISegmentedControl cambia automáticamente el tamaño de los segmentos para que se ajusten proporcionalmente a su supervista a menos que tengan un ancho específico establecido. Cuando agrega y elimina segmentos, puede solicitar que la acción sea animada con efectos de deslizamiento y desvanecimiento.

Existen los siguientes pasos para agregar control de segmentos al generador de interfaces.

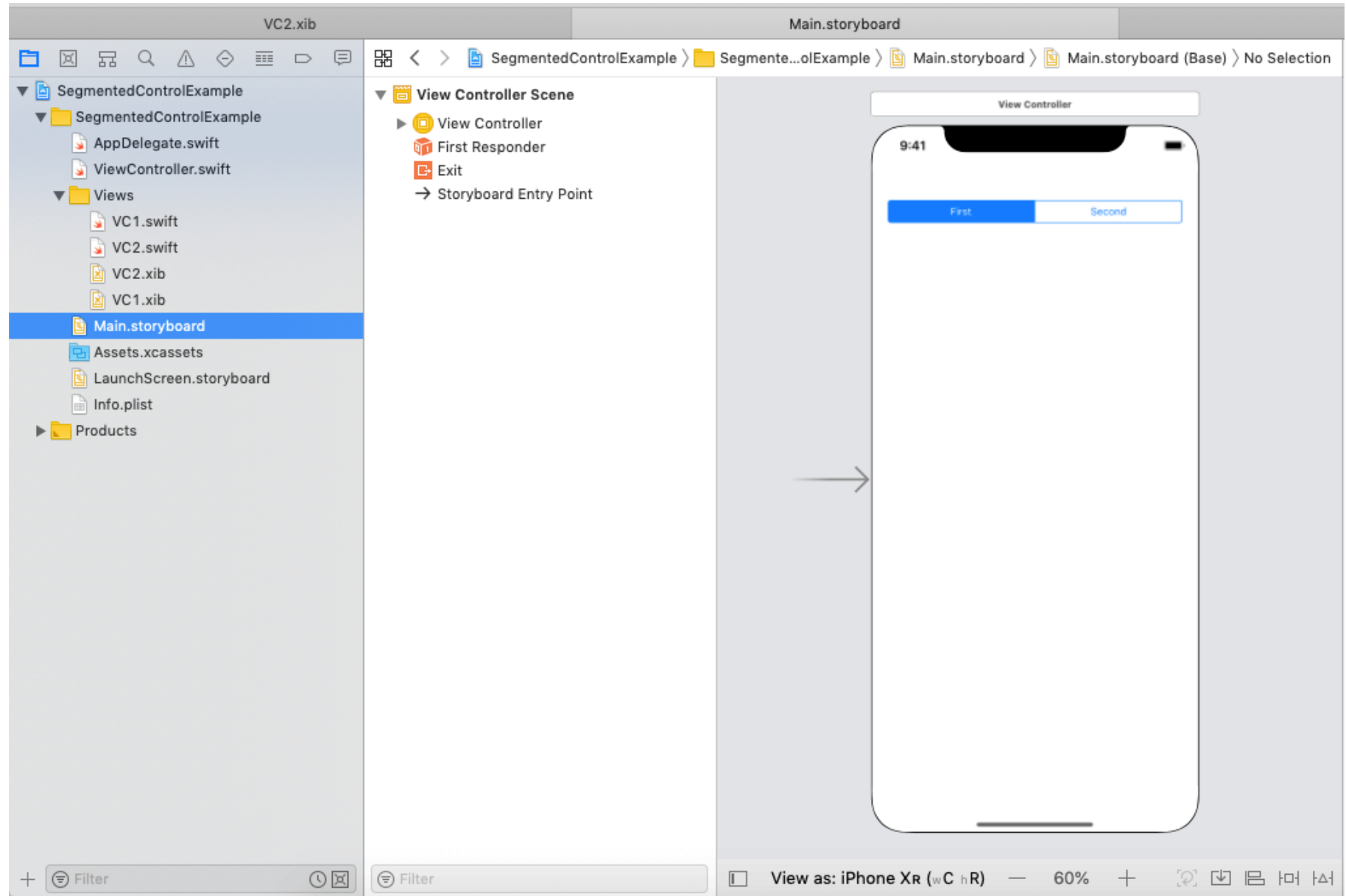
1. Busque SegmentControl en la biblioteca de objetos y arrastre el resultado al guión gráfico.
2. Cree una salida de SegmentControl para personalizar la apariencia de SegmentControl.
3. Cree una conexión de acción de SegmentControl para preparar la lógica para el evento de cambio de valor activado.
4. Configure las reglas de Diseño automático para el Control de segmento para controlar el tamaño y la posición del Control de segmento en dispositivos iOS de diferentes tamaños.

Ejemplo

En este ejemplo, agregaremos el control de segmento a nuestro generador de interfaces y usaremos ese control para mostrar los archivos XIB individuales que creamos, respectivamente.

Interface Builder

En este ejemplo, hemos utilizado dos XIB para mostrar las vistas individuales en caso de control de segmento horizontal. El guión gráfico principal contiene control de segmento, que controla cómo se muestran las vistas en consecuencia.

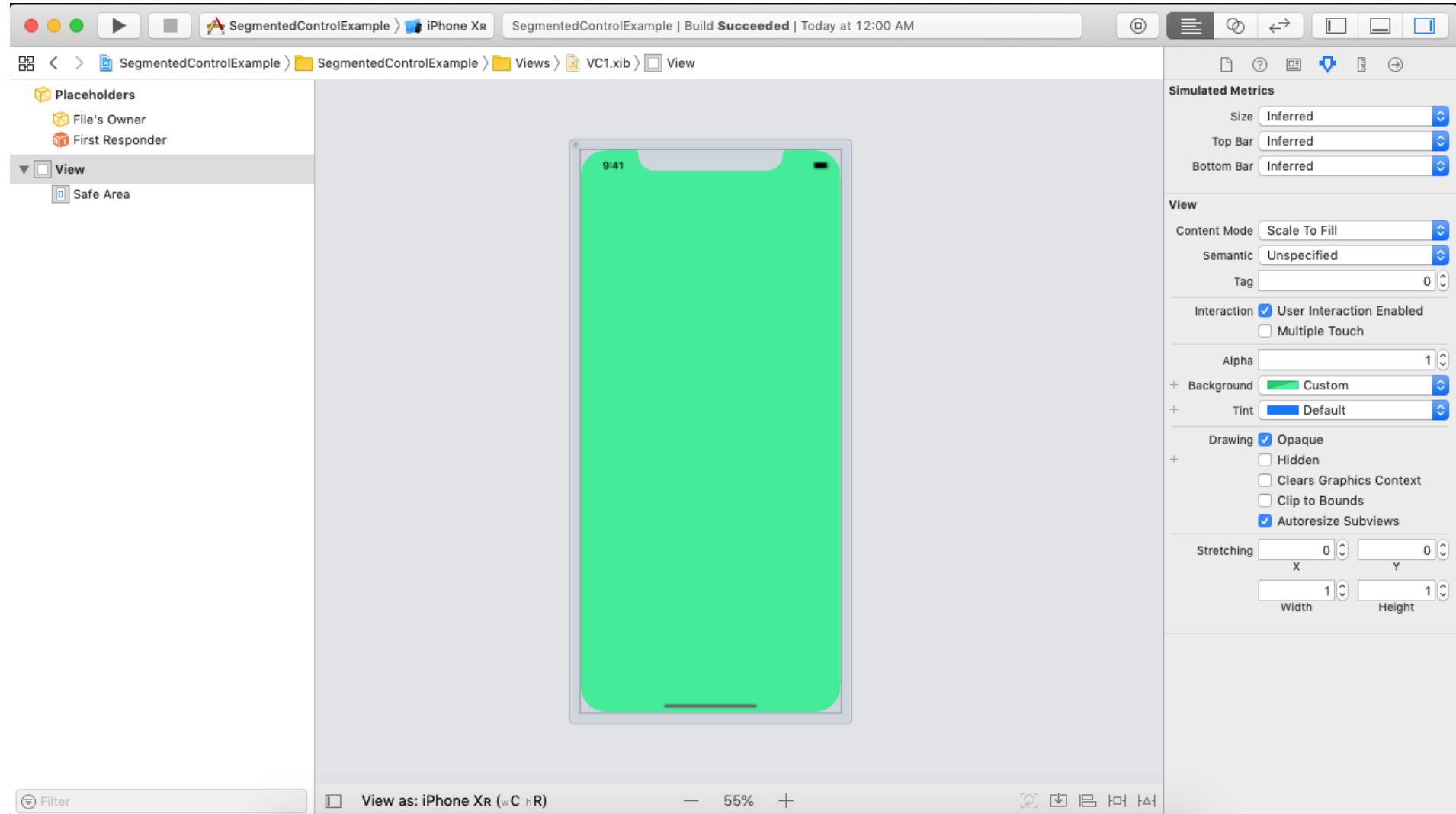


¿Qué es un archivo XIB?

XIB significa un generador de interfaz XML. El generador de interfaces nos permite desarrollar interfaces gráficas de usuario con la ayuda de cacao y carbono. Los archivos XIB se cargan en tiempo de ejecución para proporcionar la interfaz de usuario para la aplicación. Los archivos XIB se almacenan como archivos NIB o XIB, que representan UIView.

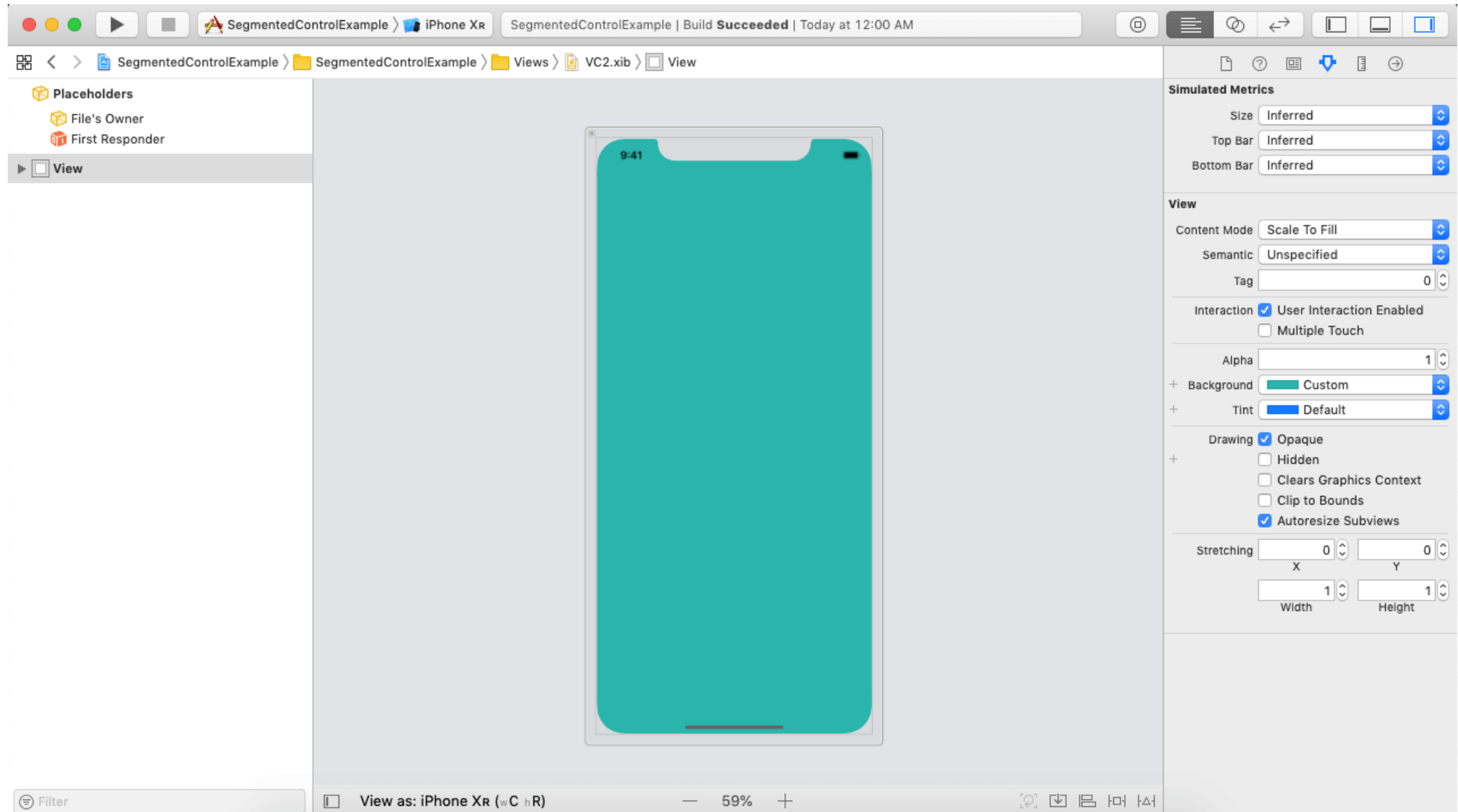
VC1.xib

La siguiente imagen muestra el archivo VC.xib.



VC2.xib

La siguiente imagen muestra el archivo VC2.xib.



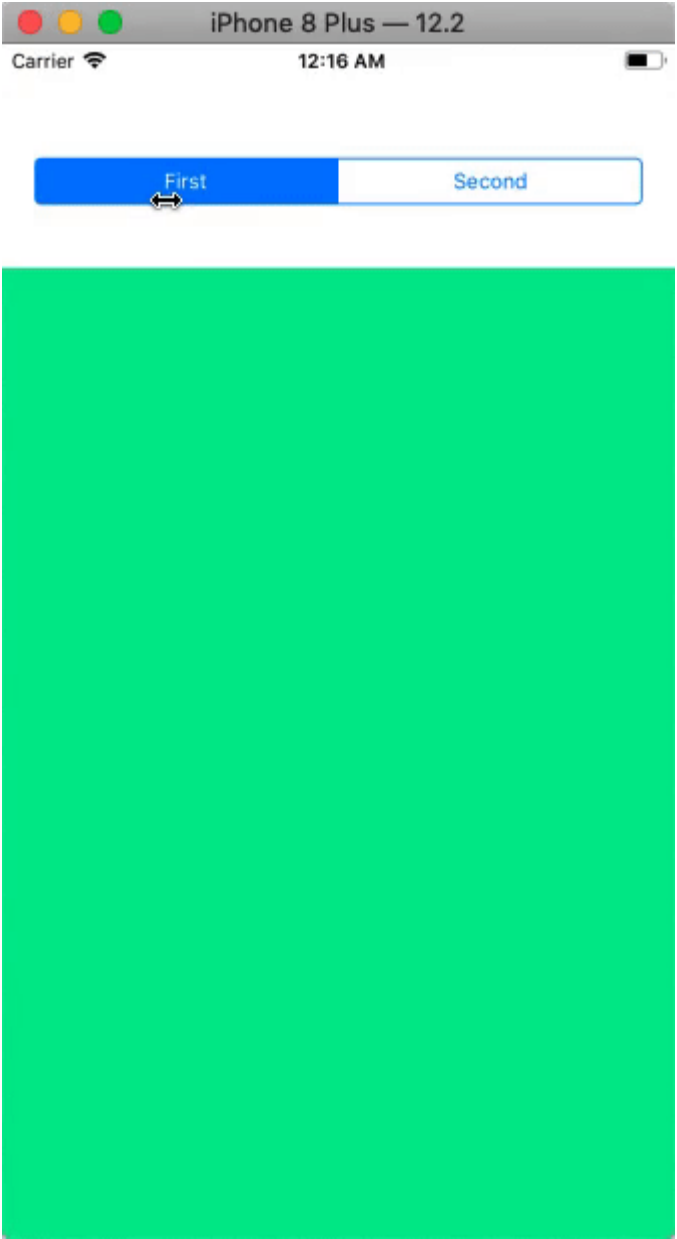
ViewController.swift

En el archivo ViewController.swift, crearemos la conexión de acción para el control de segmento, que se notifica cada vez que se desencadena el evento valueChanged para el control de segmento.

Alterna entre VC1.xib y VC2.xib en el evento valueChanged de Segment Control.

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.
6.    @IBOutlet weak var segmentedControl: UISegmentedControl!
7.
8.    @IBOutlet weak var viewContainer: UIView!
9.
10.    var views = Array<UIView>()
11.
12.    override func viewDidLoad() {
13.        super.viewDidLoad()
14.        // Do any additional setup after loading the view.
15.        views.append(VC1().view!)
16.        views.append(VC2().view!)
17.        for v in views{
18.            viewContainer.addSubview(v)
19.        }
20.        viewContainer.bringSubviewToFront(views[0])
21.    }
22.
23.    @IBAction func switchViewAction(_ sender: UISegmentedControl) {
24.        viewContainer.bringSubviewToFront(views[sender.selectedSegmentIndex])
25.    }
26.}
```

Salida:



iOS Container Views

UIView

UIView se puede definir como un objeto mediante el cual podemos crear y administrar el área rectangular en la pantalla. Podemos tener cualquier cantidad de vistas dentro de una vista para crear una estructura jerárquica de las UIViews.

La UIView se gestiona utilizando los métodos y propiedades definidos en la clase UIView que hereda UIKit. La declaración de UIView se da de la siguiente manera.

clase UIView: UIKit

Las vistas son los fundamentos del desarrollo de aplicaciones iOS, y es por eso que UIView es uno de los objetos más utilizados en la biblioteca de objetos. Las vistas son el bloque de construcción básico de la aplicación iOS, que representa el contenido dentro de sus límites rectángulo y también maneja cualquier interacción con el contenido.

Las UIViews son los fundamentos y el punto de conexión de la aplicación iOS con el usuario. Las vistas de la aplicación iOS realizan varias actividades.

- Dibujo y animación
 - Al usar vistas, podemos dibujar en el área rectangular de la pantalla.
- Diseño y gestión de subvista
 - Podemos incrustar una o más subvistas en la UIView. La apariencia de las subvistas se puede administrar administrando la apariencia de la súper vista.
 - Podemos definir las reglas de diseño automático para gobernar el tamaño y el posicionamiento de la jerarquía de vistas en diferentes dispositivos iOS.
- Manejo de eventos
 - Una vista puede responder al toque y a otro tipo de evento, ya que es la subclase de UIResponder.
 - Podemos agregar los reconocedores de gestos para la vista uiview, como UITapGestureRecognizer.

Creando UIView

Se puede crear una vista mediante programación instanciando la clase UIView. Podemos pasar el objeto frame dentro del constructor UIView. En el desarrollo de aplicaciones iOS, hay muchos objetos como etiquetas, campos de texto, etc., que heredan directamente la clase UIView para usar las propiedades y métodos comunes.

```
1.let rect = CGRect(x: 10, y: 10, width: 100, height: 100)
2.let myView = UIView(frame: rect)
```

Ejemplo 1

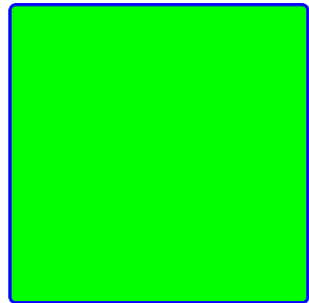
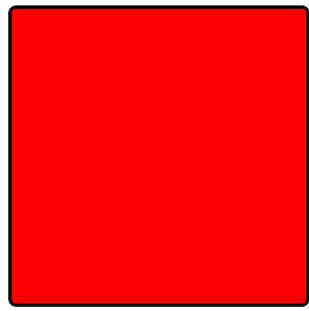
En este ejemplo, crearemos dos UIViews mediante programación y estableceremos sus propiedades simultáneamente.

ViewController.swift

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    override func viewDidLoad() {
6.        super.viewDidLoad()
7.        // Do any additional setup after loading the view.
8.        let frame1 = CGRect(x: 100, y: 100, width: 200, height: 200)
9.        let myView1 = UIView(frame: frame1)
10.        myView1.layer.shadowColor = UIColor.black.cgColor
11.        myView1.layer.borderColor = UIColor.black.cgColor
12.        myView1.layer.borderWidth = 2
13.        myView1.layer.cornerRadius = 5
14.        myView1.layer.shadowRadius = 2
15.
16.        let frame2 = CGRect(x: 100, y: 400, width: 200, height: 200)
17.        let myView2 = UIView(frame: frame2)
18.        myView2.layer.shadowColor = UIColor.blue.cgColor
19.        myView2.layer.borderColor = UIColor.blue.cgColor
20.        myView2.layer.borderWidth = 2
21.        myView2.layer.cornerRadius = 5
22.        myView2.layer.shadowRadius = 2
23.
24.        myView2.backgroundColor = .green
25.        myView1.backgroundColor = .red
26.        view.addSubview(myView1)
27.        view.addSubview(myView2)
28.    }
29.}
```

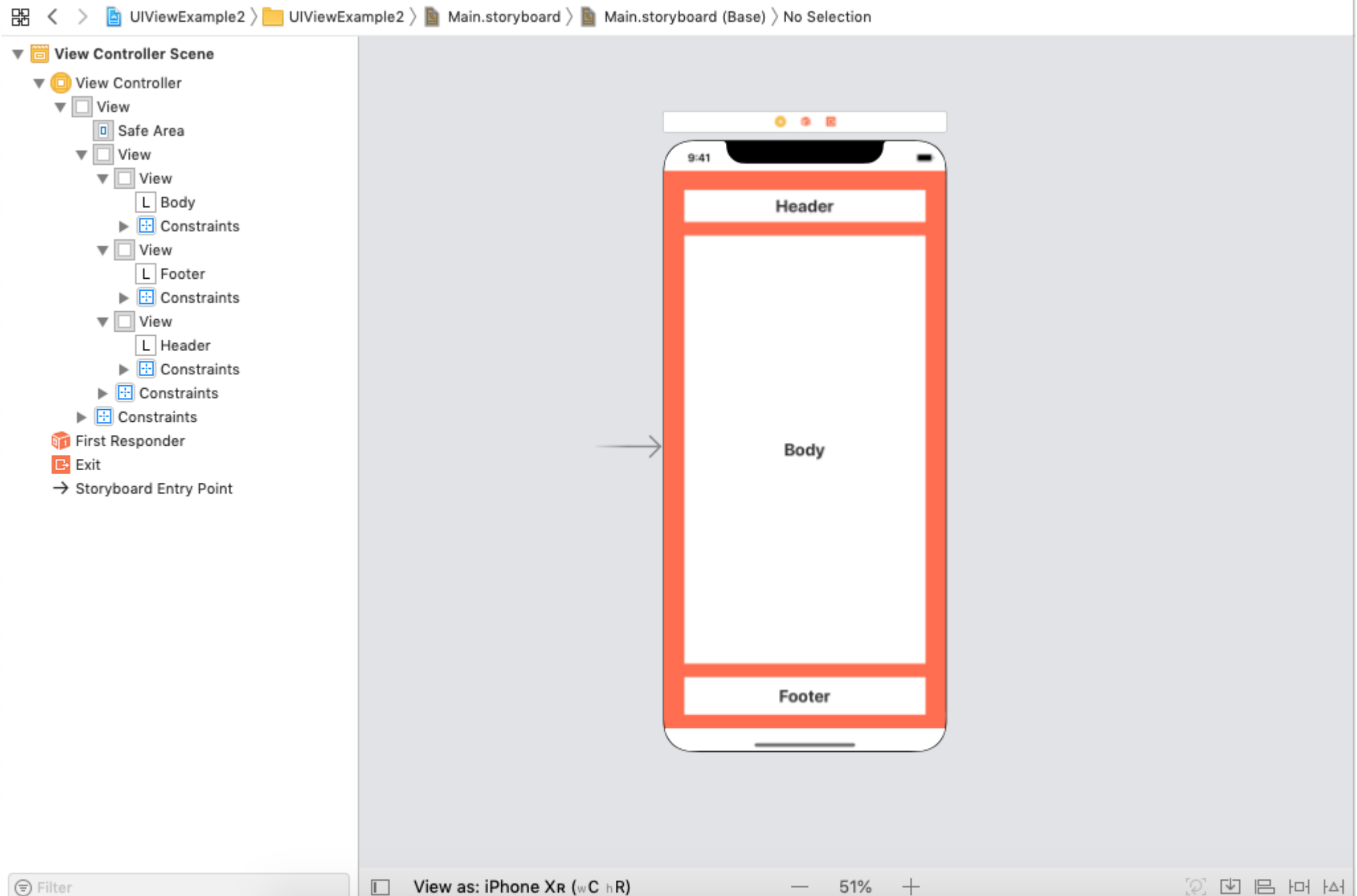
Carrier 

10:52 PM



En este ejemplo, simularemos la estructura de la página web en la aplicación iOS. En este tipo de aplicación iOS, donde necesitamos mostrar personalizaciones separadas, usaremos UIViews en las aplicaciones iOS.

La siguiente imagen muestra el generador de interfaces (`main.storyboard`) desarrollado en el proyecto. El panel izquierdo de la ventana muestra la jerarquía de vistas y etiquetas utilizadas en el proyecto.

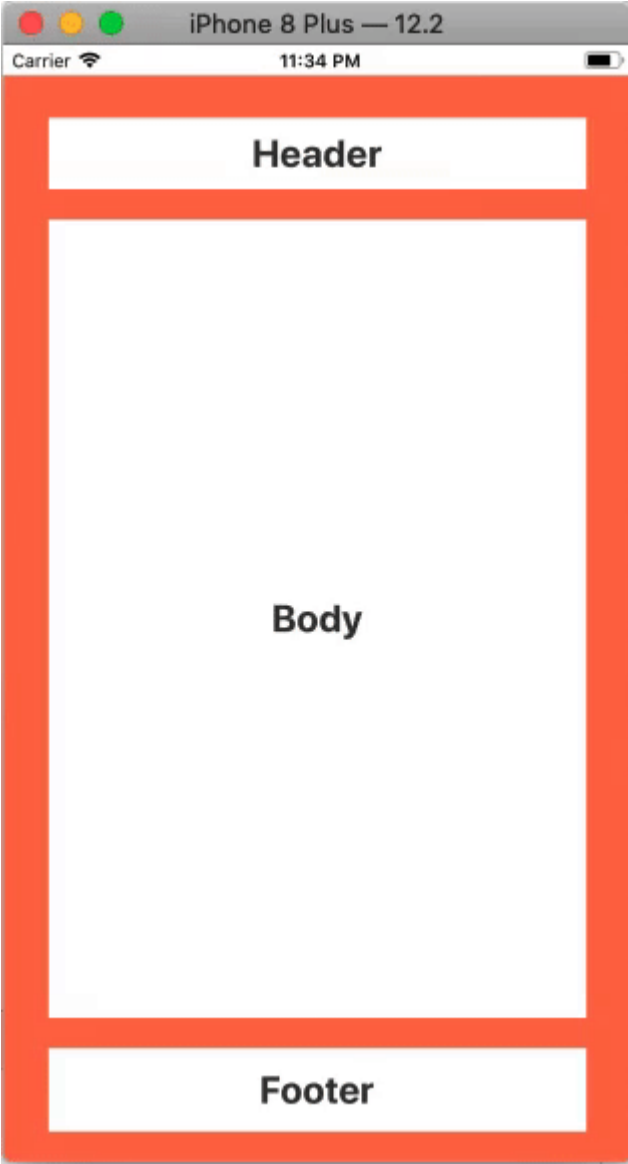


ViewController.swift

En el archivo ViewController, agregaremos el comportamiento de las etiquetas para que podamos cambiar el aspecto de la UIView asociada con la etiqueta.

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    @IBOutlet weak var headerView: UIView!
6.
7.    @IBOutlet weak var bodyView: UIView!
8.
9.    @IBOutlet weak var footerView: UIView!
10.
11.    @IBOutlet weak var headerLbl: UILabel!
12.
13.    @IBOutlet weak var bodyLbl: UILabel!
14.
15.    @IBOutlet weak var footerLbl: UILabel!
16.
17.    override func viewDidLoad() {
18.        super.viewDidLoad()
19.        // Do any additional setup after loading the view.
20.
21.        let headerTapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(headerLblTapped))
22.        headerLbl.isUserInteractionEnabled = true
23.        headerLbl.addGestureRecognizer(headerTapGestureRecognizer)
24.
25.        let bodyTapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(bodyLblTapped))
26.
27.        bodyLbl.isUserInteractionEnabled = true
28.        bodyLbl.addGestureRecognizer(bodyTapGestureRecognizer)
29.
30.        let footerTapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(footerLblTapped))
31.        footerLbl
32.        footerLbl.isUserInteractionEnabled = true
33.        footerLbl.addGestureRecognizer(footerTapGestureRecognizer)
34.    }
35.
36.    @objc func headerLblTapped(){
37.        headerView.backgroundColor = .orange
38.    }
39.
40.    @objc func bodyLblTapped(){
41.        bodyView.backgroundColor = .green
42.    }
43.
44.
45.    @objc func footerLblTapped(){
46.        footerView.backgroundColor = .orange
47.    }
48.}
```

Salida



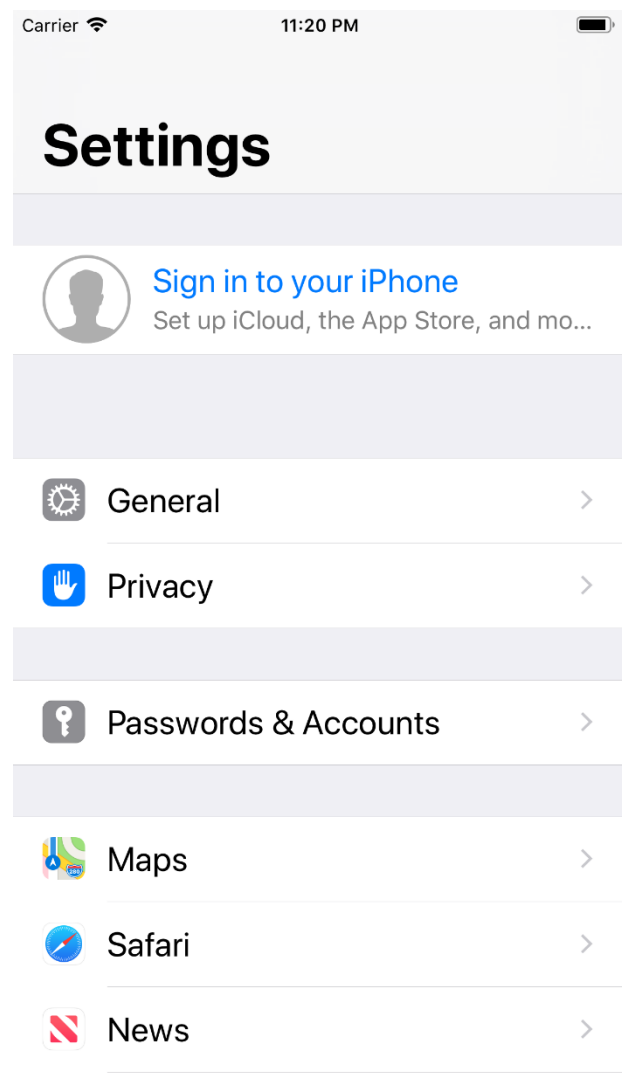
TableView

TableView se puede definir como la vista que puede organizar los datos utilizando filas en una sola columna. Se usa en casi todas las aplicaciones de iOS, por ejemplo, contactos, facebook, Instagram, etc. La vista de tabla es la instancia de la clase UITableView, que hereda la clase UIScrollView. Discutiremos UIScrollView en los próximos capítulos de este tutorial.

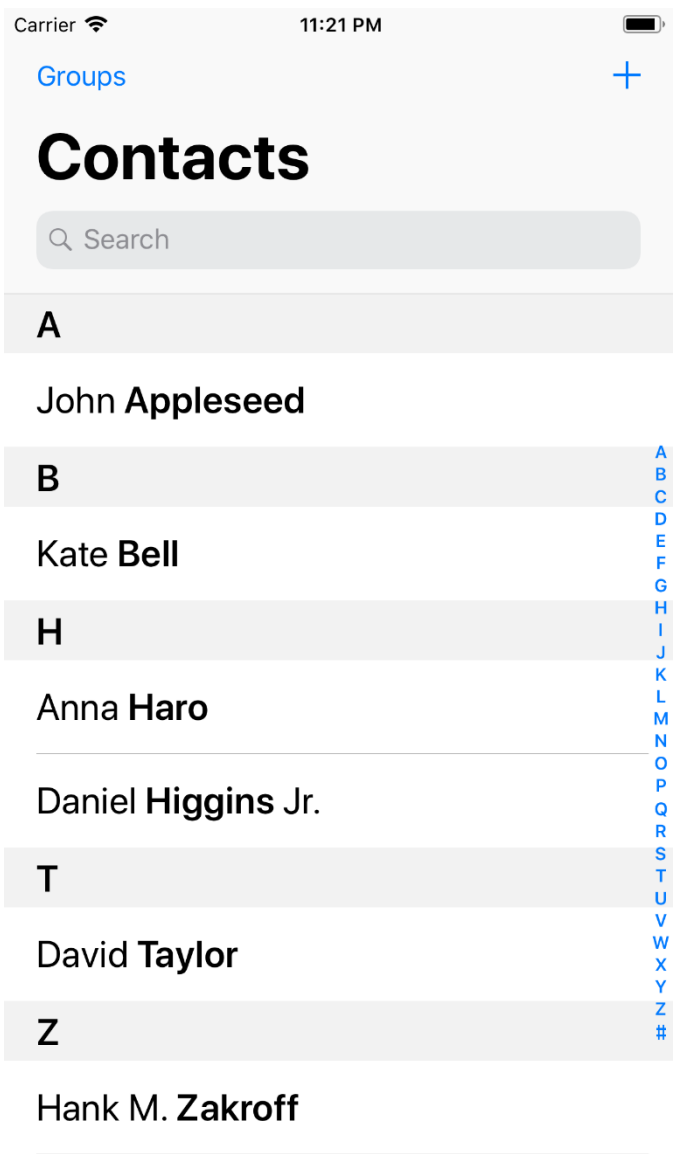
clase UITableView: UIScrollView

En las aplicaciones de iOS, siempre que necesitemos mostrar una sola columna que contenga contenido de desplazamiento vertical, usamos tableView. La vista de tabla puede mostrar múltiples registros (divididos en filas), que pueden desplazarse verticalmente si es necesario. Cada fila de la vista de tabla presenta cada registro de la fuente de datos. Por ejemplo, en la aplicación de contacto, mostramos cada nombre de contacto en la fila separada de la vista de tabla, y obtenemos los detalles relacionados con el contacto al hacer clic en esa fila.

La siguiente imagen muestra cómo se utiliza la vista de tabla para mostrar datos en la aplicación de configuración.



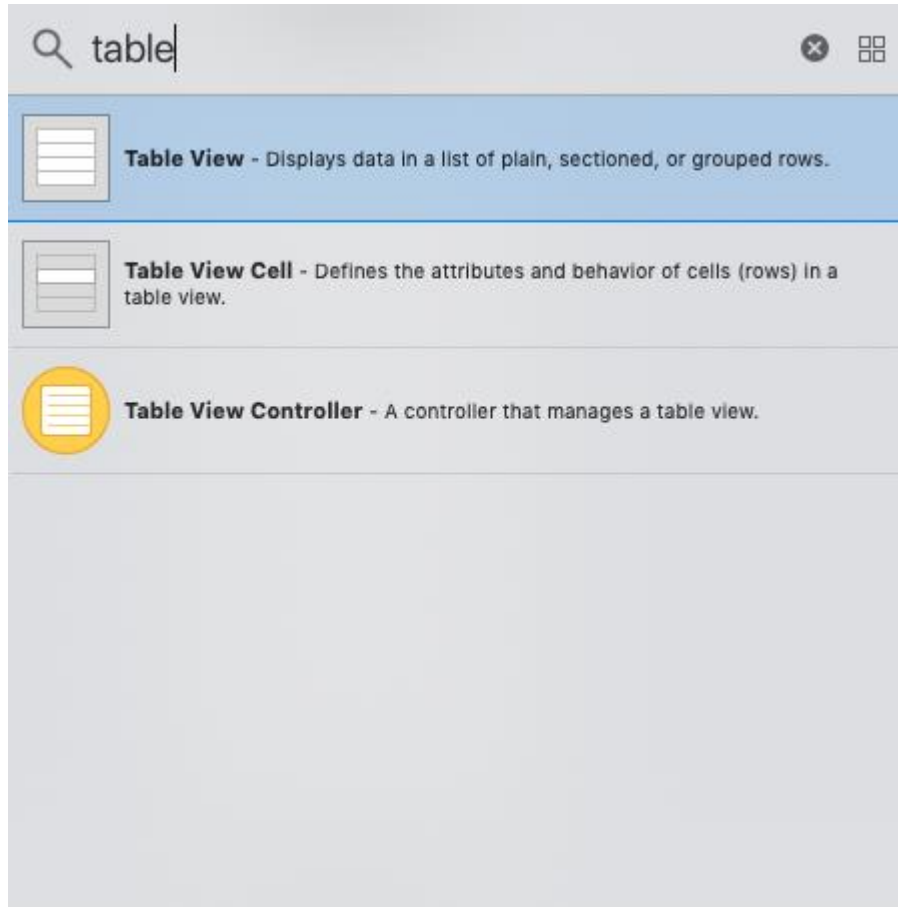
La siguiente imagen muestra cómo se usa la vista de tabla en la aplicación Contacto.



Podemos crear las secciones en la vista de tabla para agrupar las filas relacionadas, o podemos mostrar la larga lista de registros en varias filas sin secciones. En las aplicaciones de iOS, la vista de tabla se usa en asociación con el controlador de navegación para organizar los datos jerárquicamente. Aquí, podemos navegar entre diferentes niveles de jerarquía usando el controlador de navegación. La apariencia de la vista de tabla es administrada por la clase UITableView, que hereda UIScrollView. En tableview, la fila es simulada por el objeto de la clase UITableViewCell, que se puede usar para mostrar el contenido real. Podemos personalizar las celdas de vista de tabla para mostrar cualquier contenido en la aplicación iOS.

Agregar UITableView a la interfaz

Para agregar la vista de tabla al gui n gr fico, busque la vista de tabla en la biblioteca de objetos y arrastre el resultado al gui n gr fico.



Para usar la vista de tabla, necesitamos establecer sus propiedades de delegado y fuente de datos. La vista de tabla es un objeto controlado por datos, es decir, obtiene los datos que se mostrar n desde el objeto de origen de datos. En las aplicaciones del mundo real, el objeto de origen de datos contiene los datos que devuelve una llamada API desde el servidor de la base de datos.

Las propiedades de delegado y fuente de datos de la vista de tabla se pueden establecer utilizando la siguiente l nea de c digo en el m todo viewDidLoad de ViewController.

```
1.tableView.delegate = self  
2.tableView.datasource = self
```

Métodos de delegado de TableView

Los métodos de delegado de vista de tabla se definen para agregar las siguientes características a la vista de tabla.

- Podemos crear encabezados y pies de página personalizados para las secciones en la vista de tabla.
- Podemos especificar las alturas personalizadas para filas, encabezados y pies de página.
- Proporcione estimaciones de altura para las filas, encabezados y pies de página.
- Podemos definir el método que puede manejar las selecciones de fila.

SN	Método	Descripción
1	<code>func tableView (UITableView, willDisplay: UITableViewCell, forRowAt: IndexPath)</code>	La vista de tabla notifica a este delegado cuando está a punto de dibujar una celda para una fila en particular.
3	<code>func tableView (UITableView, willSelectRowAt: IndexPath) -> IndexPath?</code>	La vista de tabla notifica este método de delegado cuando la fila especificada está a punto de seleccionarse.
4 4	<code>func tableView (UITableView, didSelectRowAt: IndexPath)</code>	Este delegado recibe una notificación cuando se selecciona la fila especificada de la vista de tabla.
5 5	<code>func tableView (UITableView, willDeselectRowAt: IndexPath) -> IndexPath?</code>	Este delegado recibe una notificación cuando la celda particular está a punto de ser deseleccionada.
6 6	<code>func tableView (UITableView, didDeselectRowAt: IndexPath)</code>	Este delegado recibe una notificación cuando la fila particular no está seleccionada.
7 7	<code>func tableView (UITableView, viewForHeaderInSection: Int) -> UIView?</code>	Este método delegado devuelve una UIView que representa el encabezado de la vista de tabla.
8	<code>func tableView (UITableView, viewForFooterInSection: Int) -> UIView?</code>	Este método delegado devuelve la uiview, que representa el pie de página de la vista de tabla.
9 9	<code>func tableView (UITableView, willDisplayHeaderView: UIView, forSection: Int)</code>	Este método de delegado se notifica cuando la vista de tabla va a mostrar la vista de encabezado para la sección particular.
10	<code>func tableView (UITableView, willDisplayFooterView: UIView, forSection: Int)</code>	Este método de delegado se notifica cuando la vista de tabla va a mostrar la vista de pie de página para la sección en particular.
11	<code>func tableView (UITableView, heightForRowAt: IndexPath) -> CGFloat</code>	Este método de delegado devuelve la altura de la fila.
12	<code>func tableView (UITableView, heightForHeaderInSection: Int) -> CGFloat</code>	Este método delegado devuelve la altura del encabezado de la sección en la vista de tabla.
13	<code>func tableView (UITableView, heightForFooterInSection: Int) -> CGFloat</code>	Este método de delegado devuelve la altura del pie de página de una sección en particular en la vista de tabla.
14	<code>func tableView (UITableView, estimatedHeightForRowAt: IndexPath) -> CGFloat</code>	Pide al delegado la altura estimada de la fila en una ubicación particular.
15	<code>func tableView (UITableView, estimatedHeightForHeaderInSection: Int) -> CGFloat</code>	Pide al delegado la altura estimada del encabezado en una ubicación particular.
dieciséis	<code>func tableView (UITableView, estimatedHeightForFooterInSection: Int) -> CGFloat</code>	Pide al delegado la altura estimada para el pie de página en la sección particular.

Métodos de fuente de datos de TableView

Para mantener los datos que se mostrarán en la vista de tabla, necesitamos mantener un objeto DataSource que implemente el protocolo UITableViewDataSource. El objeto de origen de datos gestiona los datos de vista de tabla. El objeto fuente de datos realiza las siguientes tareas principales.

- 1.Informa el número de filas y secciones que se mostrarán en la vista de tabla.
- 2.Asigna las celdas reutilizables para cada fila en la vista de tabla.
- 3.Proporciona los títulos para encabezados y pies de página en las secciones de vista de tabla.

Para realizar las tareas mencionadas anteriormente, hay algunas funciones definidas en el protocolo UITableViewDataSource. La siguiente tabla contiene los métodos importantes definidos en el protocolo.

SN	Método	Descripción
1	func tableView (UITableView, numberOfRowsInSection: Int) -> Int	Este método devuelve el número de filas que se mostrarán en la sección de la vista de tabla.
2	func numberOfSections (in: UITableView) -> Int	Este método devuelve el número de secciones que se mostrarán en la vista de tabla.
3	func tableView (UITableView, cellForRowAt: IndexPath) -> UITableViewCell	Este método devuelve el objeto de un UITableViewCell, que muestra el contenido real de una fila en particular en la vista de tabla. Este método inserta la celda para una fila particular en la vista de tabla.
4 4	func tableView (UITableView, titleForHeaderInSection: Int) -> String?	Este método devuelve una cadena que representa el título del encabezado en la sección de la vista de tabla.
5 5	func tableView (UITableView, titleForFooterInSection: Int) -> String?	Este método devuelve una cadena que representa el título del pie de página en la sección de la vista de tabla.
7 7	func tableView (UITableView, canEditRowAt: IndexPath) -> Bool	Le pide a DataSource que verifique si la fila particular es editable o no.
8	func tableView (UITableView, canMoveRowAt: IndexPath) -> Bool	Le pide a DataSource que verifique si la fila particular se puede mover a otra ubicación en la vista de tabla.
9 9	func tableView (UITableView, moveRowAt: IndexPath, a: IndexPath)	Este método mueve la fila específica a otra ubicación en la vista de tabla.
10	func sectionIndexTitles (para: UITableView) -> [String]?	Devuelve la matriz de la cadena que contiene los títulos de las secciones en la vista de tabla.

Hay dos métodos que deben definirse si ViewController implementa el protocolo UITableViewDataSource, que se menciona en el siguiente código.

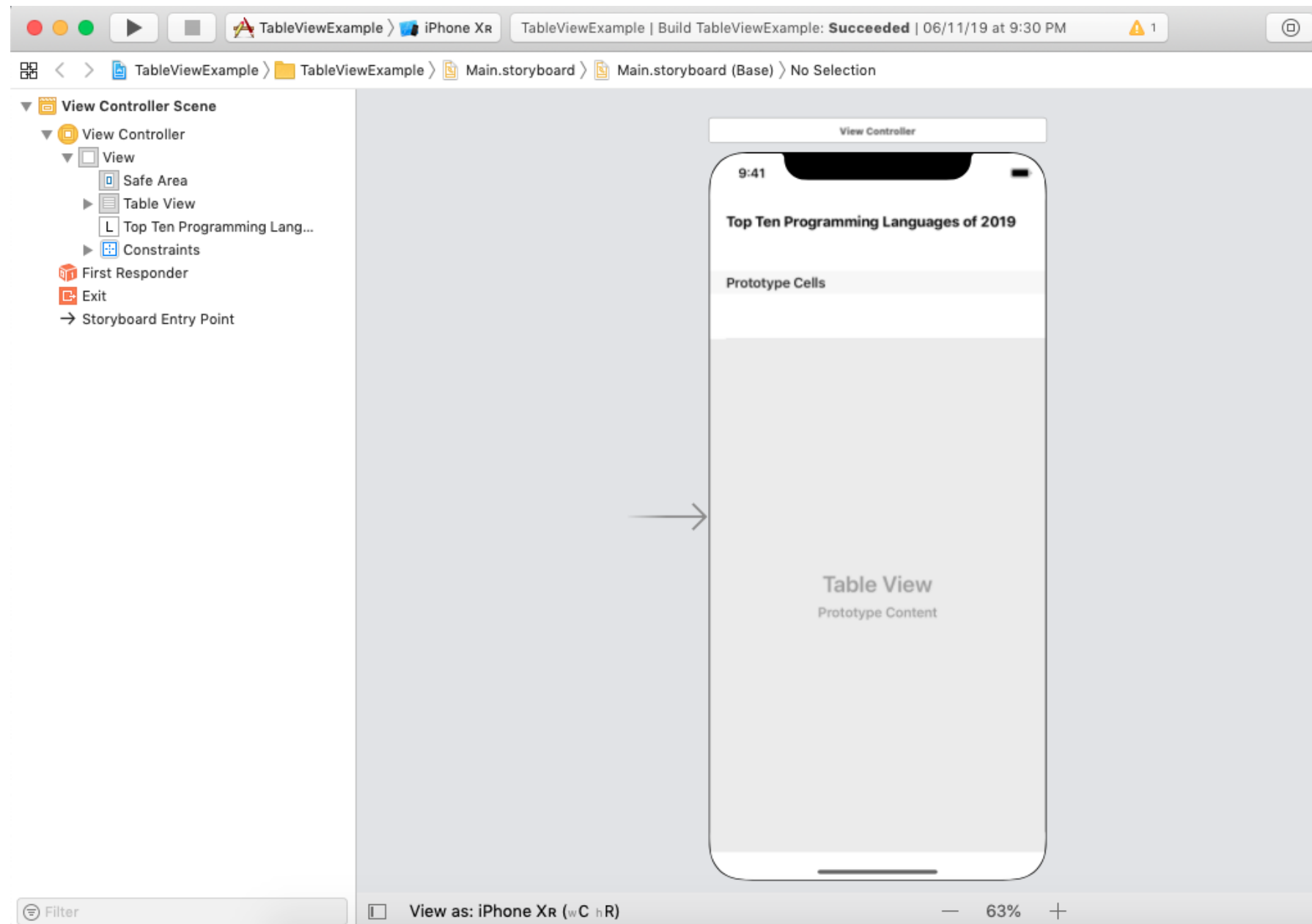
```
1.extension ViewController : UITableViewDataSource{
2.
3.    func tableView(_ tableView: UITableView, numberOfRowsInSectionSection section: Int) -> Int {
4.        return dataSourceArr.count
5.
6.    }
7.
8.    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
9.        let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath)
10.        cell.textLabel?.text = "cell text"
11.
12.        return cell
13.    }
14.
15.}
```

Ejemplo 1

En este ejemplo, crearemos una vista de tabla simple que muestra una lista de los 10 principales lenguajes de programación en 2019. En este ejemplo, usaremos UITableView para crear el generador de interfaz y los métodos de delegado y fuente de datos para establecer los datos en la vista de tabla.

Interface Builder

En este ejemplo, crearemos el siguiente controlador de vista agregando la vista de tabla al generador de interfaces. También usaremos el objeto de etiqueta para mostrar el título de la vista de tabla. Agregaremos una celda prototipo a esta vista de tabla y asignaremos la clase ViewController.swift para este ViewController.



ViewController.swift

En ViewController.swift, crearemos la salida de conexión de la vista de tabla agregada al guión gráfico. También definiremos los métodos delegado y fuente de datos para mostrar los datos de la vista de tabla.

```
1.import UIKit
2.class ViewController: UIViewController {
3.
4.
5.  @IBOutlet weak var tableView: UITableView!
6.  var dataSourceArr = Array<String>()
7.
8.  override func viewDidLoad() {
9.      super.viewDidLoad()
10.         // Do any additional setup after loading the view.
11.         tableView.delegate = self
12.         tableView.dataSource = self
13.         dataSourceArr = ["Python", "JavaScript", "Java", "Swift", "GoLang", "C#", "C++", "Scala"]
14.
15.
16.     }
17.
18.
19.}
20.
21.
22.extension ViewController : UITableViewDelegate{
23.
24.}
25.
26.
27.extension ViewController : UITableViewDataSource{
28.
29.  func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -
  > Int {
30.      return dataSourceArr.count
31.
32.  }
33.
34.  func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -
  > UITableViewCell {
35.      let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath)
36.      cell.textLabel?.text = dataSourceArr[indexPath.row]
37.      cell.textLabel?.textAlignment = .center
38.      return cell
39.  }
40.}
```


Top Ten Programming Languages of 2019

Python

JavaScript

Java

Swift

GoLang

C#

C++

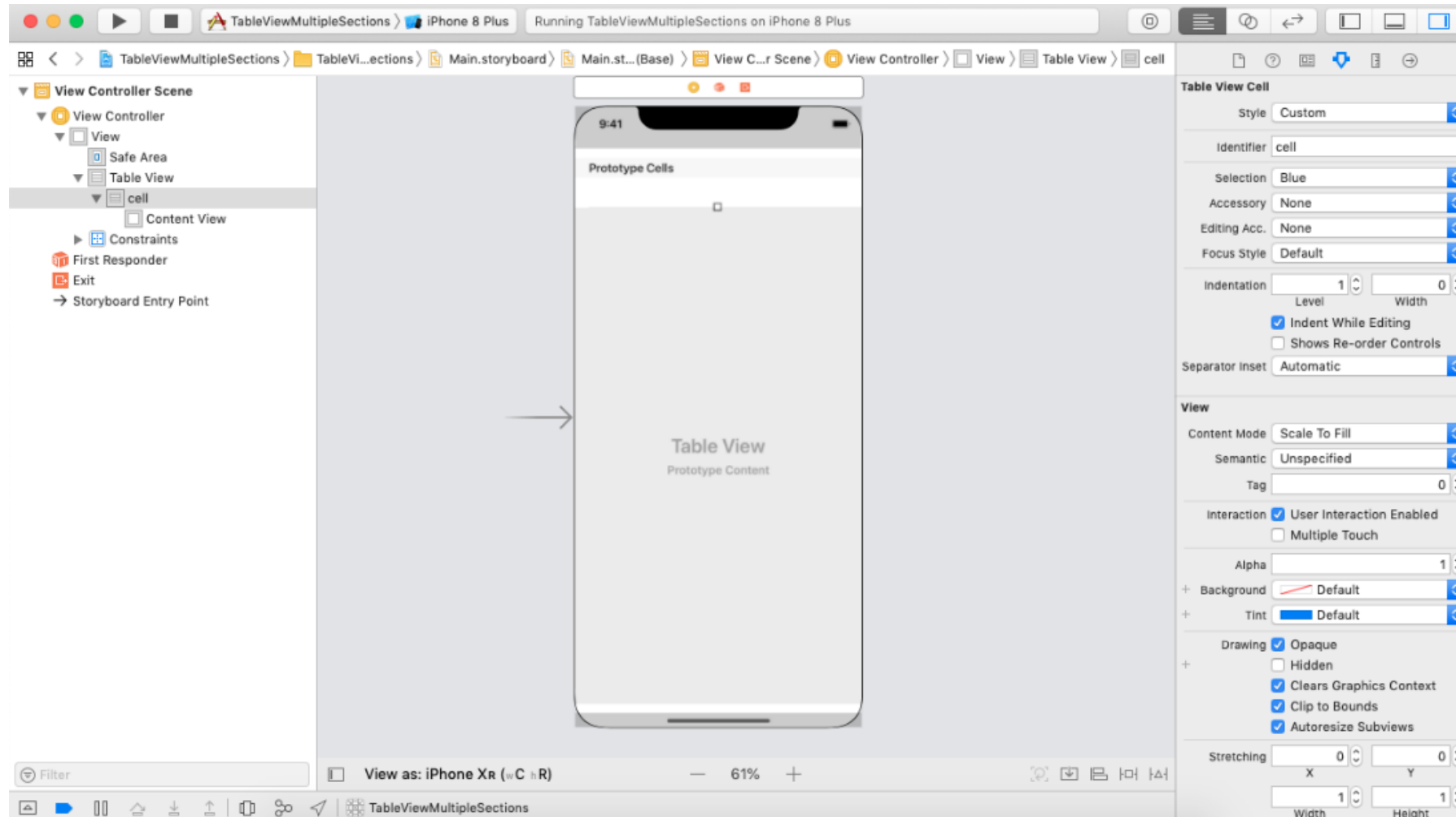
Scala

Ejemplo: manejo de múltiples secciones en la vista de tabla

En este ejemplo, crearemos las múltiples secciones en la vista de tabla y definiremos el número variable de filas y el contenido de la fila dependiendo de la sección en particular.

Interface Builder

Para crear el generador de interfaces para este ejemplo, necesitamos agregar una vista de tabla y agregar una celda prototipo para la vista de tabla. El generador de interfaces se parece a la imagen de abajo con una celda prototipo.



ViewController.swift

```
1.import UIKit
2.
3.
4.class ViewController: UIViewController {
5.
6.
7.     @IBOutlet weak var tableView: UITableView!
8.
9.     override func viewDidLoad() {
10.         super.viewDidLoad()
11.         // Do any additional setup after loading the view.
12.         tableView.delegate = self
13.         tableView.dataSource = self
14.     }
15.
16.}
17.
18.
19.extension ViewController: UITableViewDataSource{
20.
21.     func numberOfSections(in tableView: UITableView) -> Int {
22.         return 3
23.     }
24.
25.     func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
26.         if section == 0{
27.             return 2
28.         }
29.         else if section == 1{
30.             return 3
31.         }
32.         else if section == 2{
33.             return 4
34.         }
35.         return 0
36.     }
37.
38.     func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
39.         let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath)
40.         if(indexPath.section == 0){
41.             cell.textLabel?.text = "Row in section 1"
42.         }
43.         else if(indexPath.section == 1){
44.             cell.textLabel?.text = "Row in section 2"
45.         }
46.         else if(indexPath.section == 2){
47.             cell.textLabel?.text = "Row in section 3"
48.         }
49.         return cell
50.     }
51.}
52.
53.
54.extension ViewController : UITableViewDelegate{
55.
56.     func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
57.         return "Section " + (section+1).description
58.     }
59.
60.}
```

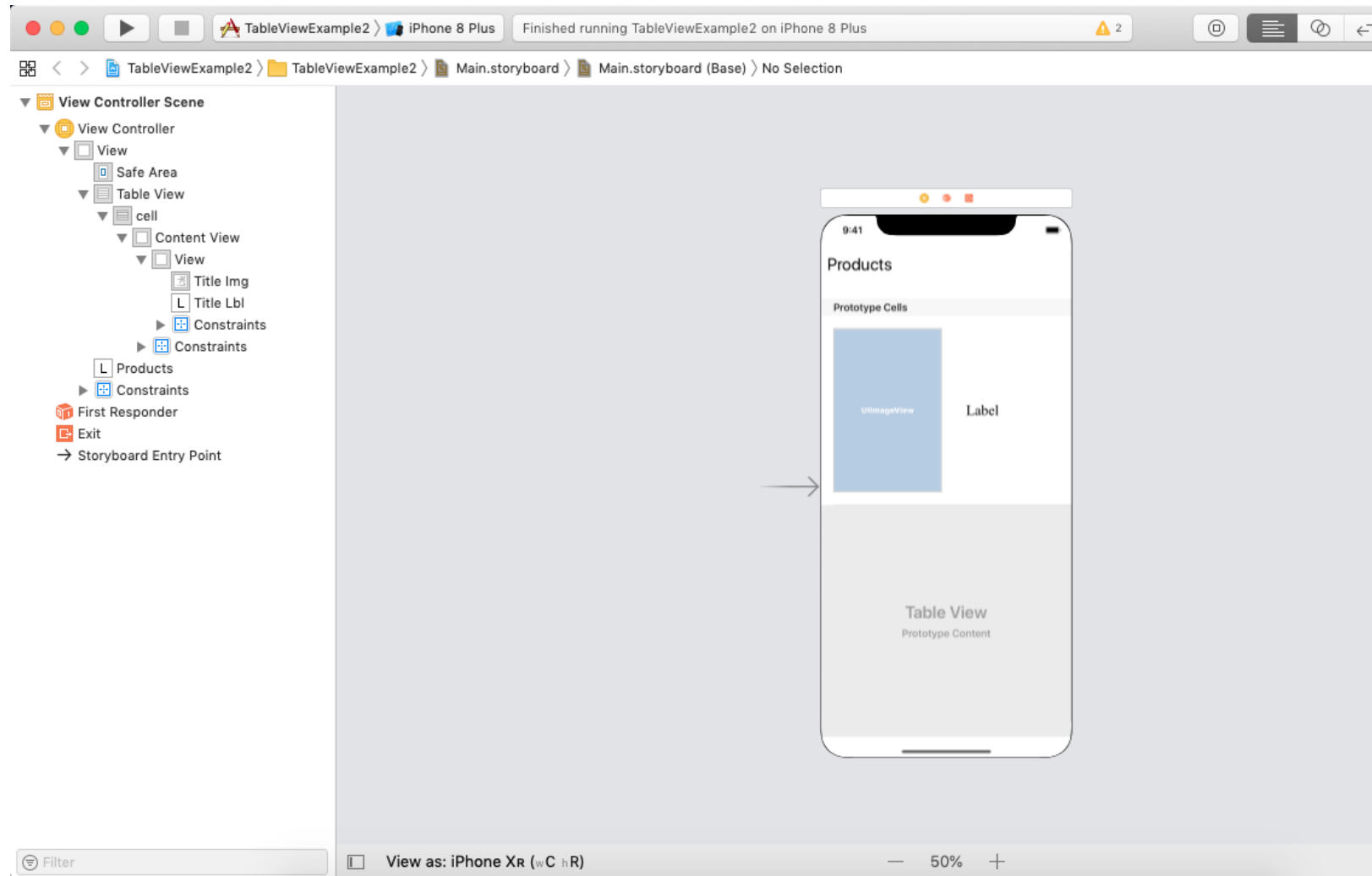
Ejemplo 2: Personalizar la celda Vista de tabla

En este ejemplo, personalizaremos la celda de vista de tabla asignándola a una clase y creando las salidas de los objetos de celda en esa clase. En la mayoría de las aplicaciones de iOS, existen los requisitos para personalizar la clase de vista de tabla, ya que no siempre podemos cumplir nuestros requisitos simplemente configurando el texto de la etiqueta de la celda.

Este ejemplo simula la vista de lista de los productos que se muestran en la aplicación de comercio electrónico.

Interface Builder

Para crear el generador de interfaces para este ejemplo, necesitamos agregar la vista de tabla al controlador de vista y agregarle la celda prototipo. En la celda prototipo de la vista de contenido, agregaremos una vista previa a la cual, agregaremos un objeto UIImageView y UILabel. La siguiente imagen muestra el guión gráfico en el ejemplo.



MyTableViewCell.swift

MyTableViewCell hereda la clase UITableViewCell, que se asigna a la celda prototipo de la vista de tabla. En esta clase, podemos instanciar la vista de imagen y etiquetar objetos.

```
1.import UIKit
2.
3.
4.class MyTableViewCell: UITableViewCell {
5.
6.
7.    @IBOutlet weak var titleImg: UIImageView!
8.
9.    @IBOutlet weak var titleLbl: UILabel!
10.
11.
12.    override func awakeFromNib() {
13.        super.awakeFromNib()
14.        // Initialization code
15.    }
16.
17.
18.    override func setSelected(_ selected: Bool, animated: Bool) {
19.        super.setSelected(selected, animated: animated)
20.
21.
22.        // Configure the view for the selected state
23.    }
24.
25.
26.}
```

ViewController.swift

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    var imgArr = ["Product1","Product2","Product3","Product4","Product5","Product6","Product7","Product8"]
6.    var lblTextArr = ["Powerbanks","Storage Devices","LED Bulbs","Laptop Bags","Keyboards","Routers","Shoes"]
7.
8.    @IBOutlet weak var tableView: UITableView!
9.
10.
11.    override func viewDidLoad() {
12.        super.viewDidLoad()
13.        // Do any additional setup after loading the view.
14.
15.        tableView.delegate = self
16.        tableView.dataSource = self
17.    }
18.
19.}
20.
21.
22.
23.
24.extension ViewController : UITableViewDataSource{
25.
26.    public func tableView(_ tableView: UITableView, numberOfRowsInSectionSection section: Int) -> Int {
27.        return imgArr.count - 1
28.    }
29.
30.    public func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
31.        let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) as! MyTableViewCell
32.        cell.titleImg?.image = UIImage(named: imgArr[indexPath.row])
33.        cell.titleLbl.text = lblTextArr[indexPath.row]
34.        return cell
35.    }
36.}
37.
38.
39.extension UIViewController : UITableViewDelegate{
40.
41.    public func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
42.        return 150
43.    }
44.
45.}
```

Products



Laptop Bags



Keyboards



Routers



Shoes

CollectionView

CollectionView es un objeto que presenta la colección ordenada de elementos de datos en los diseños personalizables. Muestra los datos en forma de un diseño de cuadrícula. Una vista de colección es una instancia de la clase UICollectionView, que hereda el UIScrollView, que se tratará más adelante en este tutorial.

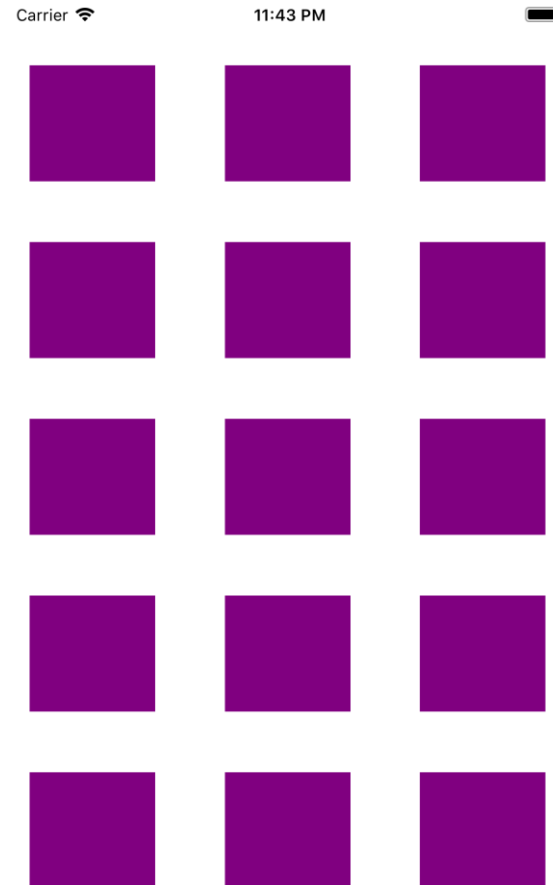
class UICollectionView: UIScrollView

Cuando agregamos la vista de colección a la interfaz de la aplicación, es responsabilidad de la aplicación iOS administrar los datos asociados con la vista de colección. El objeto CollectionView funciona de manera similar a la vista de tabla, ya que obtiene los datos del objeto DataSource, que se ajusta al protocolo UICollectionViewDataSource.

Los datos asociados con la vista de colección se tratan como elementos individuales que se pueden agrupar en las secciones para representarlos en un diseño de cuadrícula en la pantalla del iPhone. UICollectionView se usa en la mayoría de las aplicaciones de iOS, donde queremos que los datos se puedan desplazar horizontalmente.

Por ejemplo, en una aplicación de comercio electrónico como Flipkart, los productos se muestran usando UICollectionView. La interfaz de usuario muy básica que se puede desarrollar usando la vista de colección es una calculadora donde los botones individuales se tratan como elementos de vista de colección individuales que se pueden administrar usando los métodos delegado y DataSource definidos en los protocolos UICollectionViewDelegate y UICollectionViewDataSource.

UICollectionView presenta los datos en la pantalla, como se muestra en la imagen a continuación.



Agregar collectionView al generador de interfaces

1. Busque collectionView en la biblioteca de objetos y arrastre el resultado al guión gráfico.
2. Cree la salida de conexión de la vista de colección en ViewController.
3. Implemente los protocolos UICollectionViewDataSource y

UICollectionViewDelegate en ViewController y asigne el objeto delegado y fuente de datos al self.

```
1.collectionView.delegate = self  
2.collectionView.dataSource = self
```

4. Defina los métodos de delegado y fuente de datos para la vista de colección para dibujar el diseño de la vista de colección.

UICollectionViewCell

Es similar a UITableViewCell en la vista de tabla. UICollectionView presenta los datos en forma de colección ordenada de artículos. Un elemento es la unidad de datos más pequeña que se puede presentar utilizando el objeto UICollectionView. UICollectionView presenta el elemento utilizando una celda, que es una instancia de UICollectionViewCell. El método de fuente de datos configura la celda UICollectionView usando un identificador reutilizable. Una celda se configura usando UICollectionViewCell, como se indica a continuación.

```
1.func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {  
2.    let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "cell", for: indexPath) as! MyCollectionViewCell  
3.    cell.myView.backgroundColor = itemArr[indexPath.row]  
4.    return cell  
5. }
```

Vistas suplementarias

Un UICollectionView también puede presentar los datos utilizando las vistas suplementarias también. Las vistas complementarias son los encabezados y pies de página de sección que se configuran utilizando los métodos UICollectionViewDelegate. El soporte para vistas suplementarias puede definirse por el objeto de diseño de vista de colección. El objeto de diseño de vista de colección también define la ubicación de esas vistas.

Métodos de Delegado de collectionView

Los métodos collectionView Delegate manejan la selección, deselección, resaltando los elementos en la vista de colección. Todos los métodos de este protocolo son opcionales. La siguiente tabla contiene los métodos de delegado de collectionView más utilizados.

SN	Método	Descripción
1	func collectionView (UICollectionView, shouldSelectItemAt: IndexPath) -> Bool	Este método le pide al delegado que seleccione el elemento en una ruta de índice.
2	func collectionView (UICollectionView, didSelectItemAt: IndexPath)	Este método le dice al delegado que seleccione el elemento en una ruta de índice. Este método se ejecuta cuando se selecciona el elemento en una ruta de índice en la vista de colección.
3	func collectionView (UICollectionView, shouldDeselectItemAt: IndexPath) -> Bool	Este método le pide al delegado que anule la selección del elemento en una ruta de índice.
4 4	func collectionView (UICollectionView, didDeselectItemAt: IndexPath)	Este método se ejecuta cuando el elemento en un indexPath en la vista de colección no está seleccionado.
5 5	func collectionView (UICollectionView, shouldBeginMultipleSelectionInteractionAt: IndexPath) -> Bool	Le pregunta al delegado si los elementos múltiples se pueden seleccionar usando el gesto de panorámica de dos dedos en la vista de colección.
6 6	func collectionView (UICollectionView, didBeginMultipleSelectionInteractionAt: IndexPath)	Este método se ejecuta cuando los elementos múltiples se seleccionan usando el gesto de desplazamiento de dos dedos.
7 7	func collectionView (UICollectionView, shouldHighlightItemAt: IndexPath) -> Bool	Luego le pregunta si delega para resaltar el elemento durante el seguimiento.
8	func collectionView (UICollectionView, didHighlightItemAt: IndexPath)	Este método se ejecuta cuando el elemento se resalta en la vista de colección en alguna ruta de índice.

Métodos de fuente de datos de collectionView

La fuente de datos collectionView adopta el protocolo UICollectionViewDataSource. El objeto de origen de datos de la vista de colección es responsable de proporcionar los datos que requiere la vista de colección. Es como el modelo de datos de la aplicación collectionView. Pasa los datos a la vista de colección que se mostrará.

La siguiente tabla contiene los métodos declarados en el protocolo UICollectionViewDataSource.

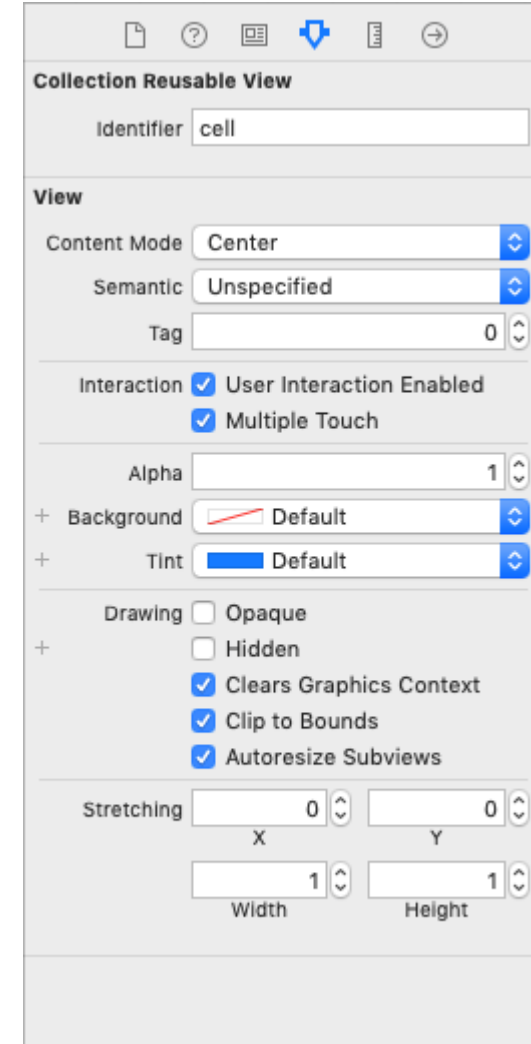
SN	Método	Descripción
1	func collectionView (UICollectionView, numberOfItemsInSection: Int) -> Int	Este método devuelve un número entero que representa el número de elementos en la sección que se mostrarán en la vista de colección. Es el recuento de la matriz de datos asociados con la vista de colección.
2	func numberOfSections (en: UICollectionView) -> Int	Este método devuelve un número entero que representa el número de secciones que se mostrarán en la vista de colección.
3	func collectionView (UICollectionView, cellForItemAt: IndexPath) -> UICollectionViewCell	Este método devuelve la instancia de UICollectionViewCell que está configurada para mostrar el contenido real. Este método no es opcional y debe definirse si el controlador de vista cumple con el protocolo UICollectionViewDataSource.
4 4	func collectionView (UICollectionView, viewForSupplementaryElementOfKind: String, at: IndexPath) -> UICollectionViewReusableView	Este método solicita al objeto de origen de datos que proporcione una vista complementaria para mostrar en la vista de recopilación.
5 5	func collectionView (UICollectionView, canMoveItemAt: IndexPath) -> Bool	Pregunta al objeto de origen de datos si el elemento especificado se puede mover a otra ubicación en la vista de colección.
6 6	func collectionView (UICollectionView, moveItemAt: IndexPath, a: IndexPath)	Este método mueve el elemento especificado al indexPath especificado.
7 7	func indexTitles (para: UICollectionView) -> [String]?	Este método devuelve la matriz de la cadena que contiene los títulos de los índices.
8	func collectionView (UICollectionView, indexPathForIndexTitle: String, at: Int) -> IndexPath	Pide al objeto de origen de datos que devuelva la ruta del índice de un elemento de vista de recopilación que corresponde a una de las entradas de índice.

Ejemplo 1

En este ejemplo, presentaremos una cuadrícula de elementos en la pantalla usando una vista de colección.

Interface Builder (main.storyboard)

Para agregar la vista de colección al guión gráfico del proyecto, busque la vista de colección en la biblioteca de objetos y arrastre el resultado al generador de interfaces. Esto agregará la vista de colección a la interfaz. Ahora, defina las reglas de diseño automático para la vista de colección para gobernar su tamaño y posición en los diferentes dispositivos de pantalla. La celda de la vista de colección muestra el contenido real de UICollectionView. Debemos definir los atributos para la celda en el inspector de atributos del Xcode. En este ejemplo, agregamos la etiqueta a la celda de vista de colección y establecemos el texto para la etiqueta en el método de fuente de datos en el objeto de celda. Los atributos dados a la celda de vista de colección se muestran en la siguiente imagen.



Los atributos dados a la vista de colección se muestran en la siguiente imagen.

Collection View

Items

1

Layout

Flow

Scroll Direction

Vertical

Accessories

☐

Section Header

☐

Section Footer

Prefetch

☒ Prefetching Enabled

Drag and Drop

☐ Spring Loaded

Scroll View

Indicators

Default Style

☒ Show Horizontal Indicator

☒ Show Vertical Indicator

Scrolling

☒ Scrolling Enabled

☐ Paging Enabled

☐ Direction Lock Enabled

Bounce

☒ Bounce On Scroll

☒ Bounce On Zoom

☐ Bounce Horizontally

☐ Bounce Vertically

Zoom

1

1

Min

Max

Content Touch

☒ Delay Touch Down

☒ Can Cancel On Scroll

Keyboard

Do not dismiss

View

Content Mode

Scale To Fill

Semantic

Unspecified

Tag

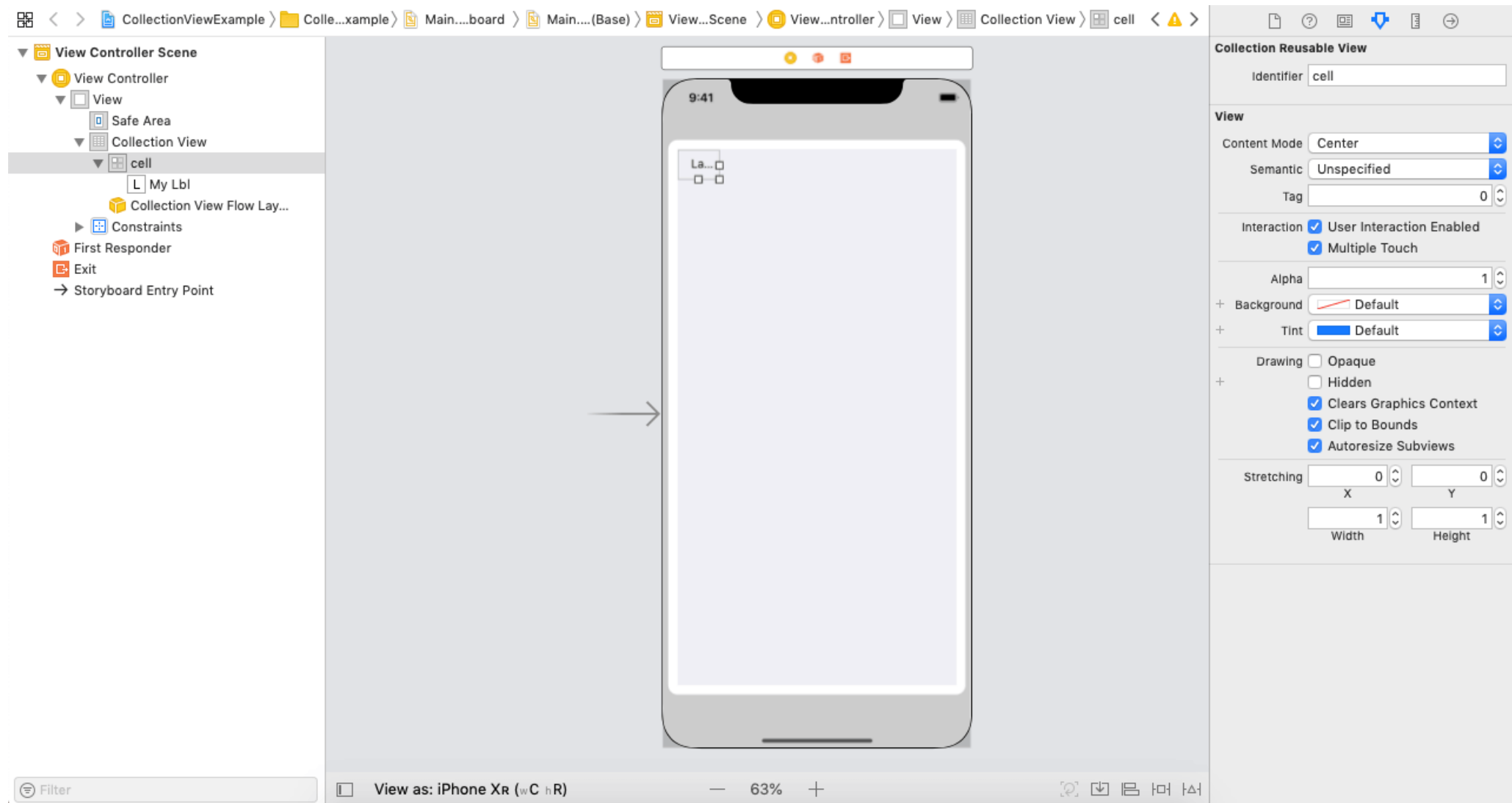
0

Interaction

☒ User Interaction Enabled

☒ Multiple Touch

El generador de interfaces creado para el proyecto se muestra a continuación.




ViewController.swift

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.
6.    @IBOutlet weak var collectionView: UICollectionView!
7.
8.    var itemArr = Array<String>()
9.
10.    override func viewDidLoad() {
11.        super.viewDidLoad()
12.        // Do any additional setup after loading the view.
13.        collectionView.delegate = self
14.        collectionView.dataSource = self
15.        for i in 1...60{
16.            itemArr.append(i.description)
17.        }
18.    }
19.}
20.
21.
22.extension ViewController : UICollectionViewDelegate{
23.
24.}
25.
26.
27.extension ViewController : UICollectionViewDataSource{
28.
29.    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -
    > Int {
30.        return itemArr.count
31.    }
32.
33.    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -
    > UICollectionViewCell {
34.        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "cell", for: indexPath) as!
        MyCollectionViewCell
35.
36.        cell.myLbl.text = itemArr[indexPath.item]
37.        cell.myLbl.layer.borderColor = UIColor.blue.cgColor
38.        cell.myLbl.textAlignment = .center
39.        cell.myLbl.layer.cornerRadius = 10
40.        cell.myLbl.layer.borderWidth = 2
41.
42.        return cell
43.    }
44.}
```

MyCollectionViewCell.swift

```
1.import UIKit
2.
3.
4.class MyCollectionViewCell: UICollectionViewCell {
5.
6.    @IBOutlet weak var myLbl: UILabel!
7.
8.}
```

Salida:

Carrier 

12:59 AM



1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36
37	38	39	40	41	42
43	44	45	46	47	48
49	50	51	52	53	54
55	56	57	58	59	60

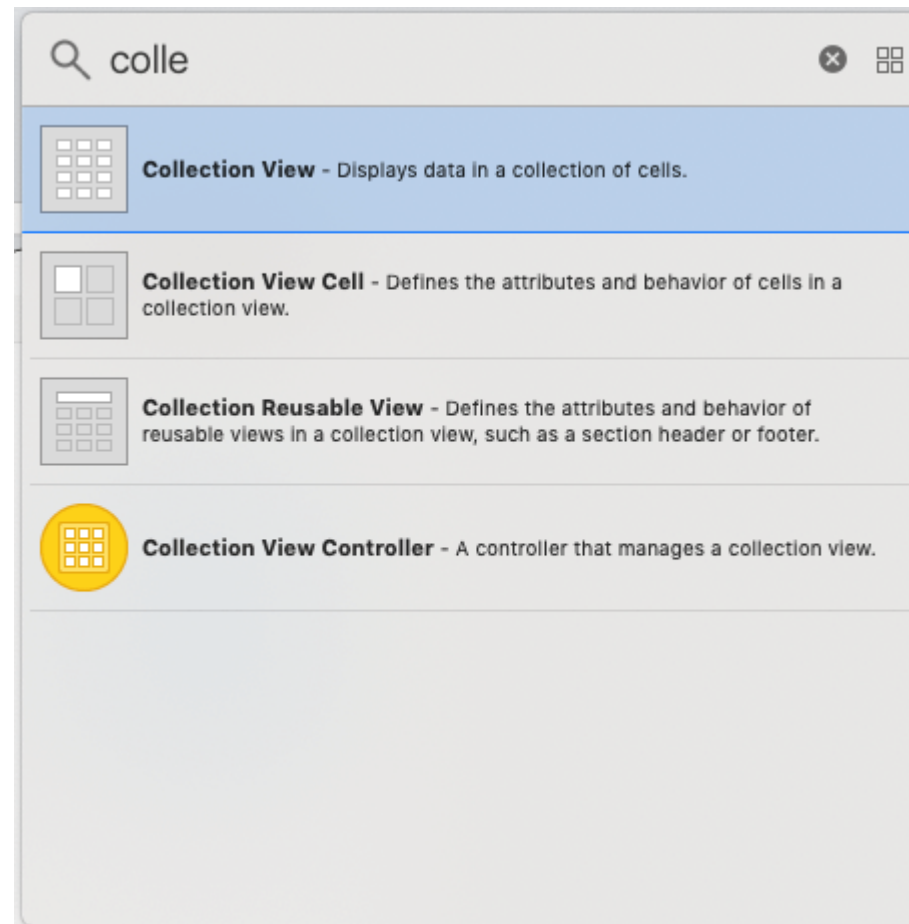
Proyecto de demostración de CollectionView

Este proyecto muestra cómo se usará la vista de colección en aplicaciones iOS. Esta aplicación simula una aplicación de comercio electrónico que muestra múltiples productos en una escena de controlador de vista. Al hacer clic en un producto, navega a otra pantalla, que muestra la vista detallada del producto específico.

En esta aplicación, utilizaremos dos controladores de vista e integraremos los controladores de vista en el controlador de navegación, que proporciona navegación a través de los controladores de vista. Le recomendamos que tenga los conocimientos básicos de los controladores de navegación, que se tratan en este tutorial.

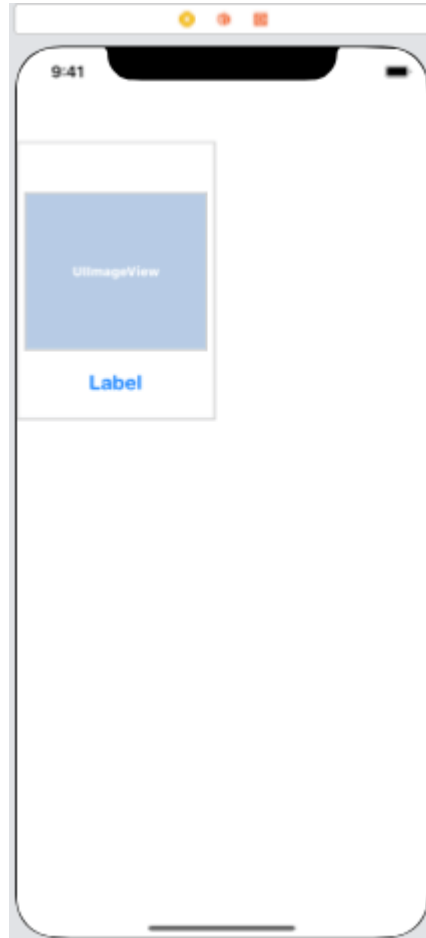
Interface Builder (main.storyboard)

Utilizaremos una vista de colección en este proyecto para mostrar una escena de cuadrícula en la aplicación. Para agregar una vista de colección, búsquela en la biblioteca de objetos y arrastre el resultado al guión gráfico.



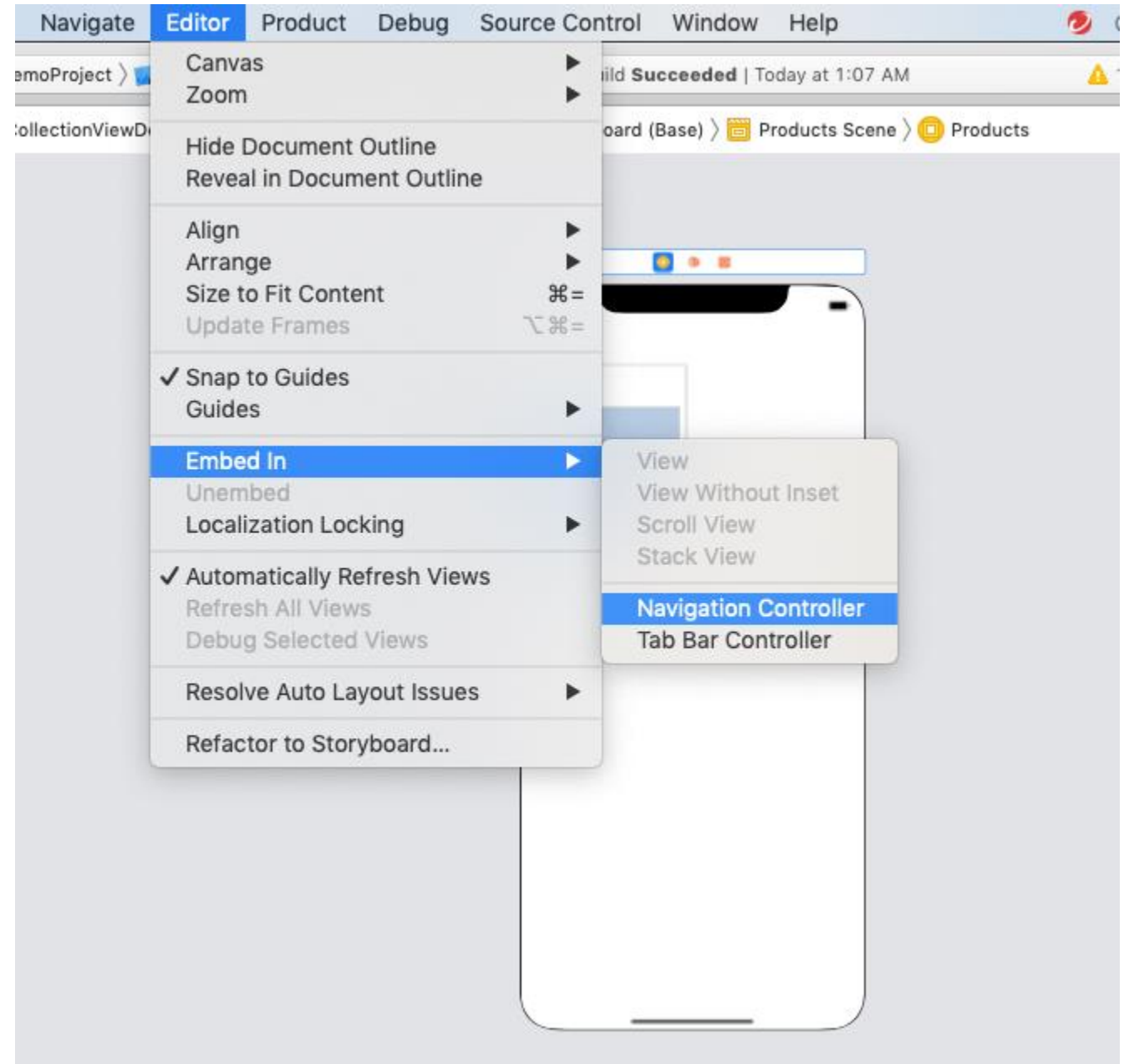
Configurar la celda

Ahora, necesitamos configurar la celda prototipo que mostrará el elemento real en la pantalla. La celda prototipo contendrá la vista de imagen para mostrar la imagen del producto y la etiqueta para mostrar el nombre de la imagen. Agregue la vista de imagen y la etiqueta a la celda de la vista de colección y asigne la clase `DemoCollectionViewCell.swift` a la celda para crear las salidas de conexión de la vista de imagen y la etiqueta respectivamente.

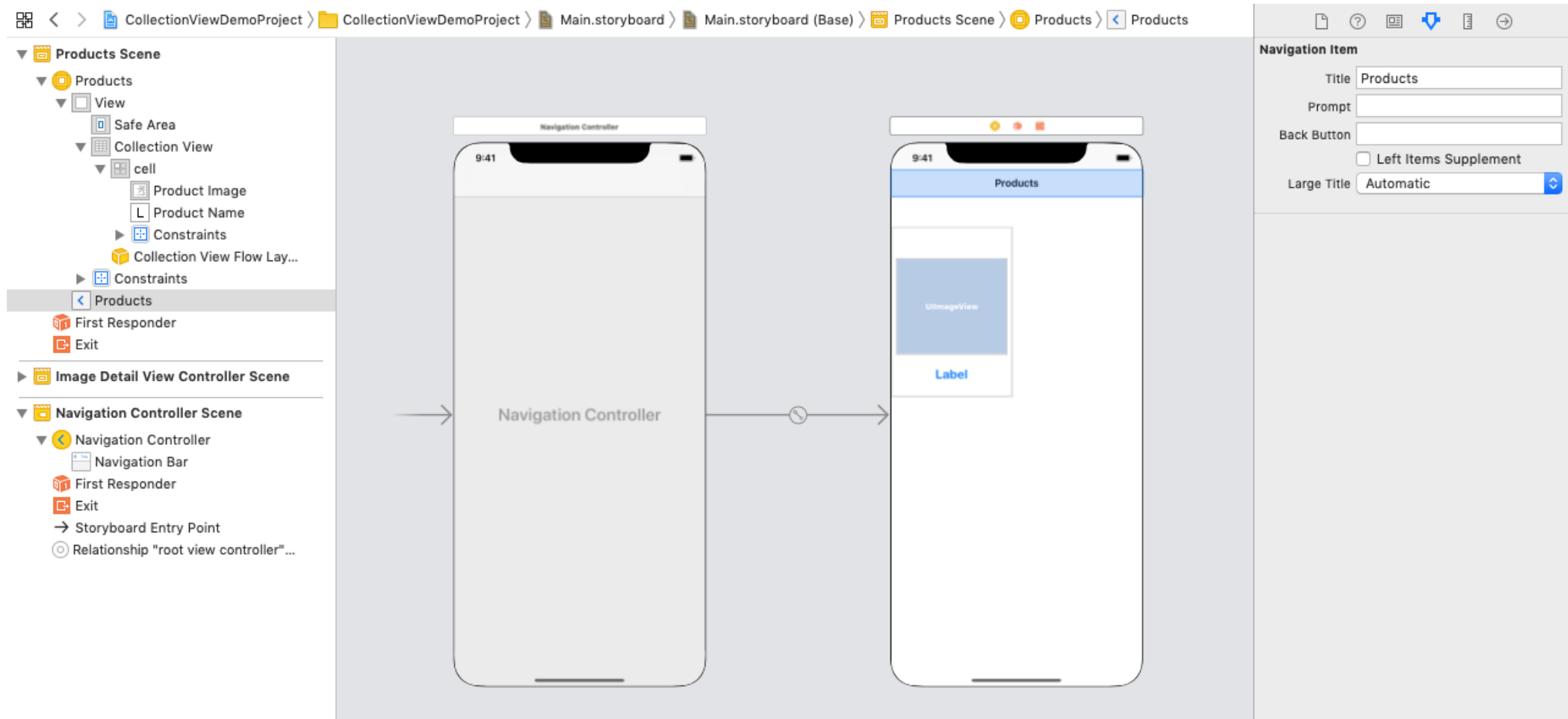


Agregar el controlador de navegación

Para incrustar el ViewController en un controlador de navegación, seleccionaremos el controlador de vista y haremos clic en el editor y elegiremos la opción **incrustar** como se muestra en la siguiente imagen.



Cuando incorporamos nuestros Controladores de vista al controlador de navegación, hay una barra de navegación que se mostrará en todos los controladores de vista de forma predeterminada. Podemos establecer ese título de la barra de navegación. Estudiaremos más sobre los controladores de navegación cuando analicemos la Interfaz de navegación en este tutorial. La siguiente imagen muestra el controlador de navegación y el controlador de vista para el que hemos establecido el título de la barra de navegación como Productos.



Agregar ImageDetailViewController

Para el generador de interfaces, necesitamos un ViewController, que puede mostrar los detalles sobre los productos. Para este propósito, agregaremos ViewController al generador de interfaces y asignaremos la clase ImageDetailViewController a este ViewController.

Al imageDetailViewController, agregaremos el UIImageView y la etiqueta al ViewController. También incluiremos este controlador de vista en nuestro controlador de navegación y configuraremos el título de la barra de navegación en Detalles del producto.

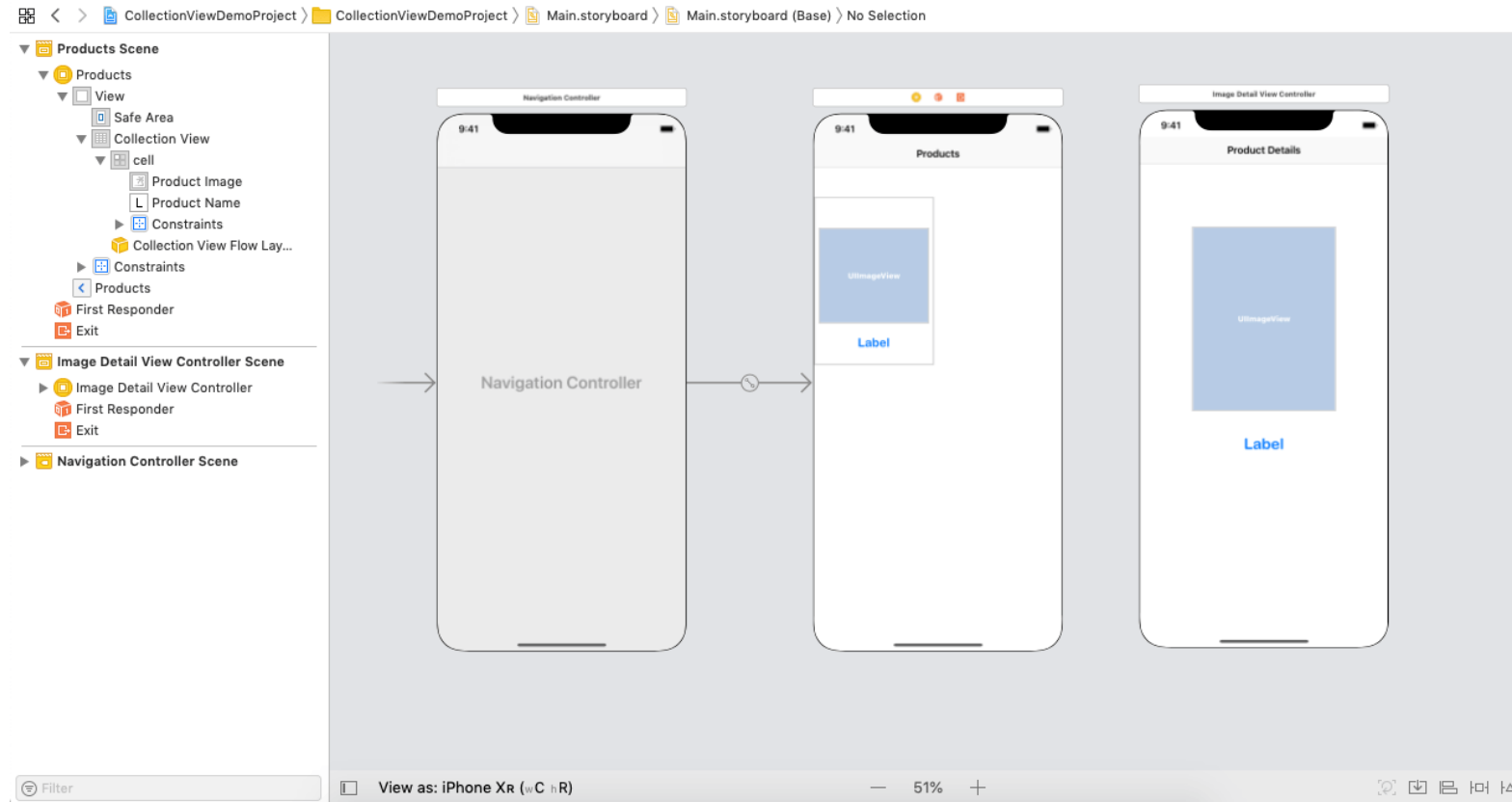


El ImageDetailViewController se mostrará cuando se seleccione un elemento en el CollectionView. Escribiremos el siguiente código en el método del origen de datos de collectionView para navegar a este controlador de vista.

```
1.let imageDetailVC = self.storyboard?.instantiateViewController(withIdentifier: "ImageDetailViewController") as! ImageDetailViewController
2.
3.    imageDetailVC.img = imgArr[indexPath.row]
4.    imageDetailVC.name = lblArr[indexPath.row]
5.    self.navigationController?.pushViewController(imageDetailVC, animated: true)
```

Cuando un controlador de navegación empuja a algún controlador de vista en el gui3n gr3fico, hay un bot3n de retroceso habilitado en ese controlador de vista, que navega hacia atr3s en el gui3n gr3fico.

El generador de interfaces construido en el proyecto de demostraci3n se muestra en la siguiente imagen.



ViewController.swift

```
1.import UIKit
2.
3.
4.class ViewController: UIViewController {
5.
6.
7.     var imgArr = Array<UIImage>()
8.     var lblArr = Array<String>()
9.
10.
11.     @IBOutlet weak var collectionView: UICollectionView!
12.
13.     override func viewDidLoad() {
14.         super.viewDidLoad()
15.         // Do any additional setup after loading the view.
16.         collectionView.delegate = self
17.         collectionView.dataSource = self
18.         for i in 1...10{
19.             lblArr.append("Watch"+i.description)
20.         }
21.         imgArr = [ imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1"), imageLiteral(resourceName: "watch1")]
22.     }
23.
24.}
25.
26.
27.extension ViewController:UICollectionViewDataSource{
28.     func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
29.         return lblArr.count
30.     }
31.
32.     func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
33.         let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "cell", for: indexPath) as! DemoCollectionViewCell
34.         cell.productImage.image = imgArr[indexPath.row]
35.         cell.productName.text = lblArr[indexPath.row]
36.         return cell
37.     }
38.
39.     func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {
40.         let imageDetailVC = self.storyboard?.instantiateViewController(withIdentifier: "ImageDetailViewController") as! ImageDetailViewController
41.
42.         imageDetailVC.img = imgArr[indexPath.row]
43.         imageDetailVC.name = lblArr[indexPath.row]
44.         self.navigationController?.pushViewController(imageDetailVC, animated: true)
45.     }
46.}
47.
48.
49.extension ViewController:UICollectionViewDelegate{
50.
51.}
```

ImageDetailViewController

```
1.import UIKit
2.
3.
4.class ImageDetailViewController: UIViewController {
5.
6.    @IBOutlet weak var productImage: UIImageView!
7.
8.    @IBOutlet weak var productName: UILabel!
9.
10.    var img:UIImage!
11.    var name:String!
12.
13.    override func viewDidLoad() {
14.        super.viewDidLoad()
15.
16.        // Do any additional setup after loading the view.
17.        productName.text = name
18.        productImage.image = img
19.    }
20.}
```

DemoCollectionViewCell.swift

```
1.import UIKit
2.
3.
4.class DemoCollectionViewCell: UICollectionViewCell {
5.
6.    @IBOutlet weak var productImage: UIImageView!
7.
8.    @IBOutlet weak var productName: UILabel!
9.}
```

Salida:



ScrollView

En aplicaciones iOS, a veces, es posible que necesitemos mostrar el contenido que no cabe en la pantalla. Para mostrar este tipo de contenido, utilizamos ScrollView en la aplicación. Un ScrollView le permite al usuario arrastrar el área del contenido. Es una instancia de la clase UIScrollView, que hereda UIView.

clase UIScrollView: UIView

ScrollView permite el desplazamiento y el zoom de sus vistas contenidas. La vista de tabla y la vista de colección son las subclases de UIScrollView y, por lo tanto, estas clases proporcionan una excelente manera de mostrar el contenido que es más grande que la pantalla. En las aplicaciones de iOS, usamos tableView para mostrar el contenido desplazable verticalmente, y la vista de colección para mostrar el contenido desplazable horizontalmente.

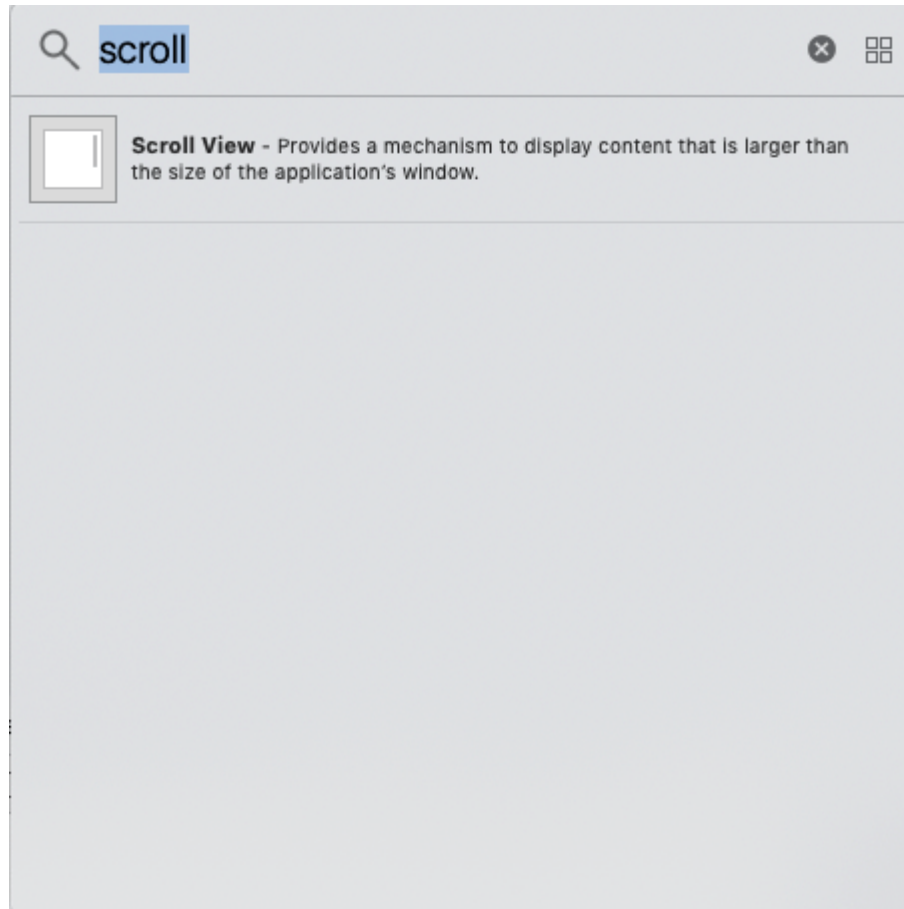
El origen del objeto UIScrollView es ajustable sobre la vista de contenido. Un ScrollView rastrea el movimiento de los dedos y ajusta su origen en consecuencia.

Agregar ScrollView a la interfaz

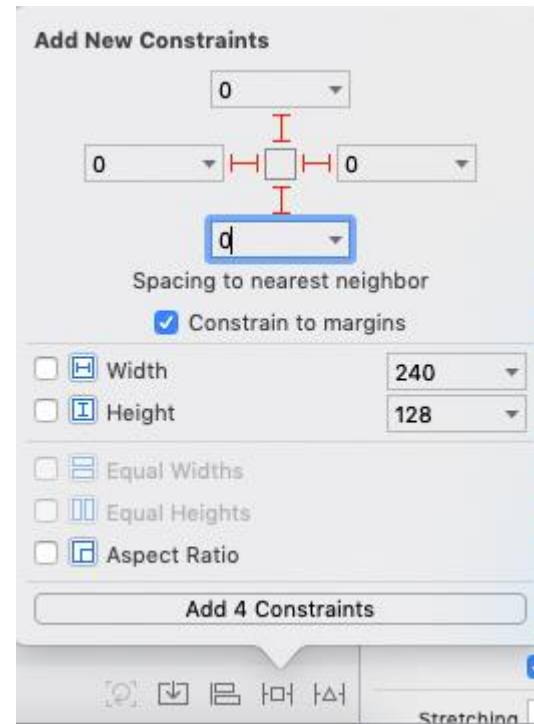
Agregar ScrollView al generador de interfaces es un poco complejo en comparación con los otros objetos. Necesitamos definir las reglas de diseño automático para guiar el diseño y la posición de la vista de desplazamiento en los diferentes dispositivos de pantalla.

Siga los siguientes pasos para agregar la vista de desplazamiento a la interfaz utilizando el generador de interfaces (main.storyboard).

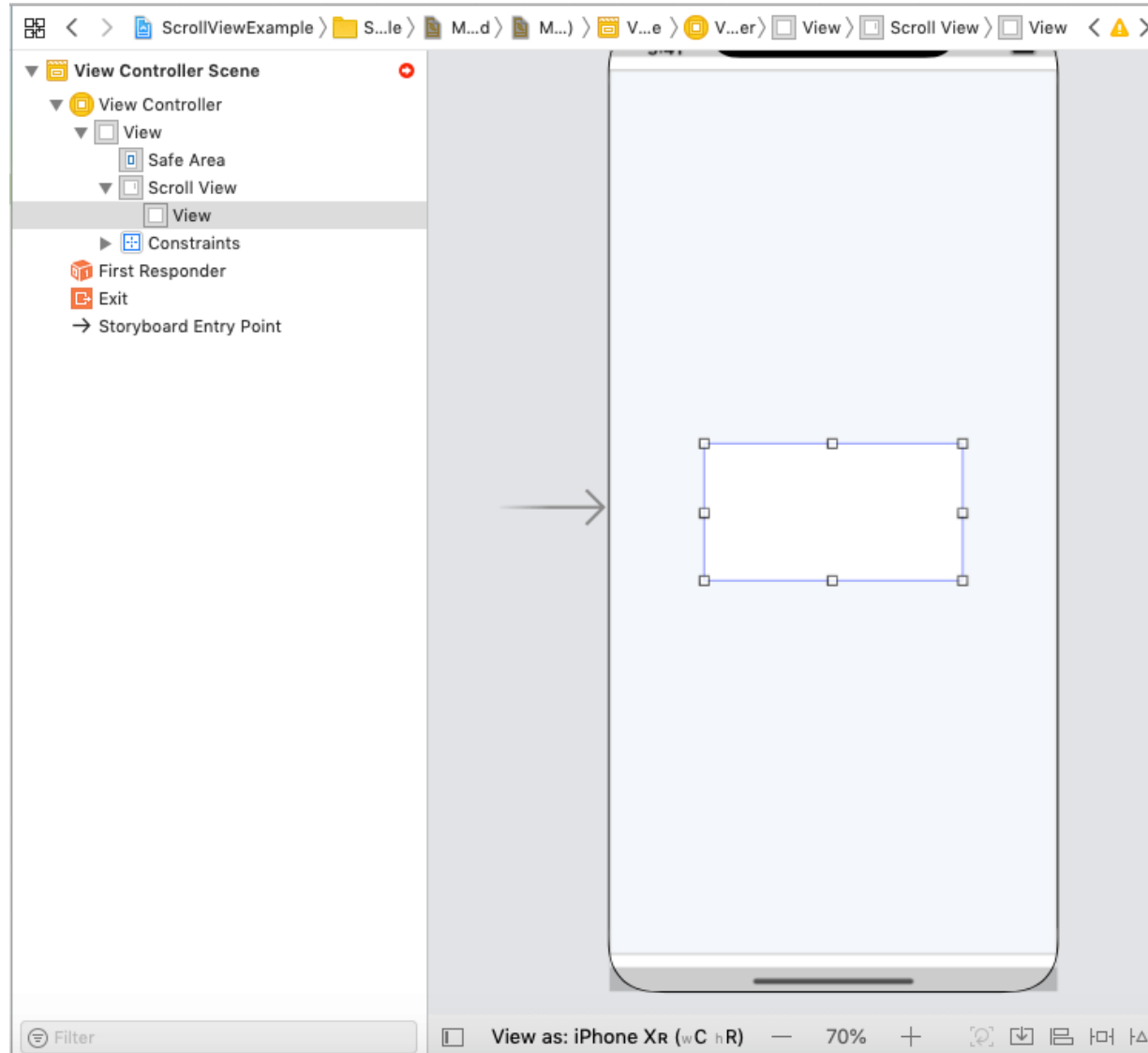
- Busque ScrollView en el main.storyboard y arrastre el resultado al ViewController.



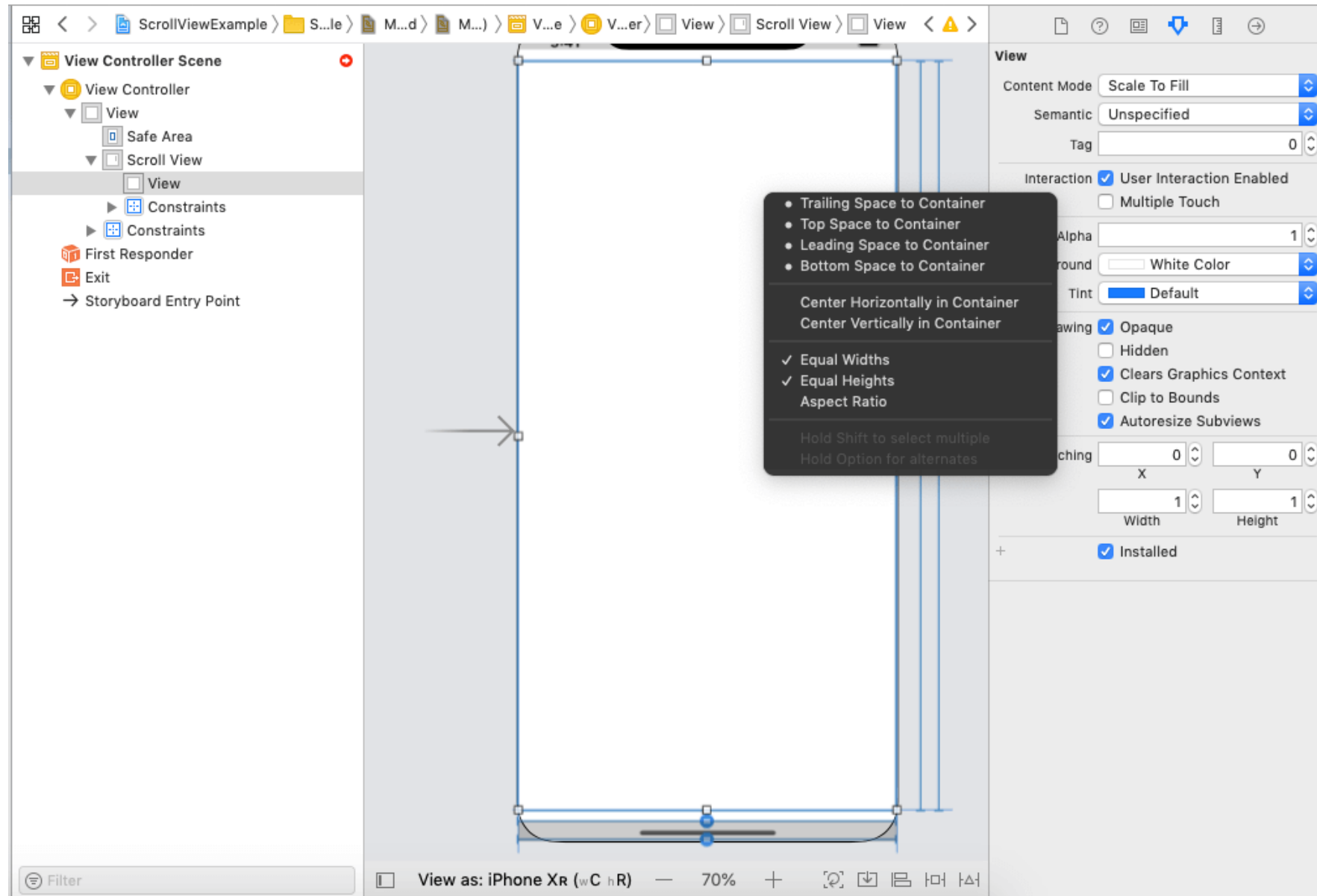
Defina las reglas de diseño automático (restricciones) para ScrollView, como se muestra en la siguiente imagen.



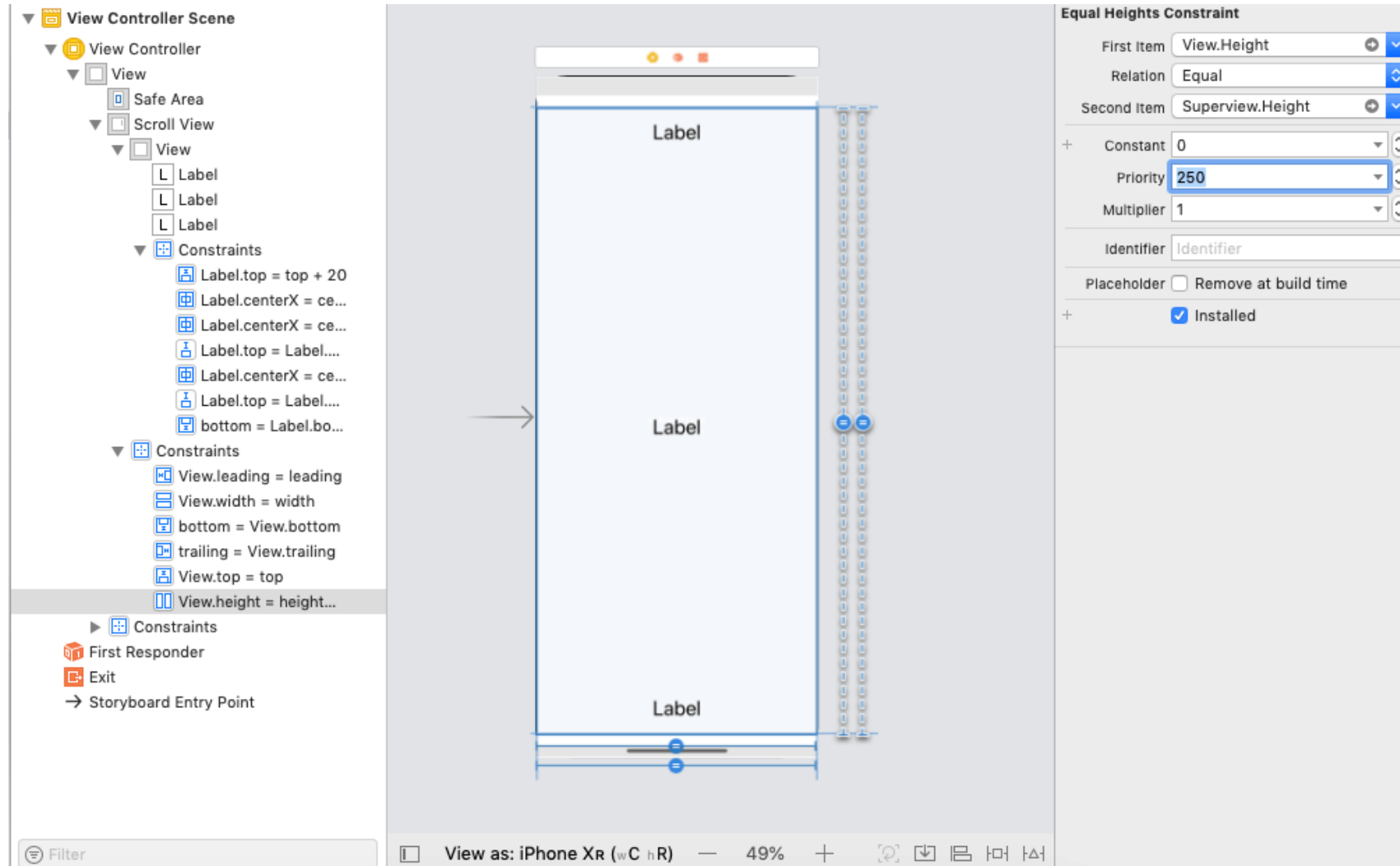
Agregue ContentView a ScrollView y defina las reglas de diseño automático para la vista de contenido.



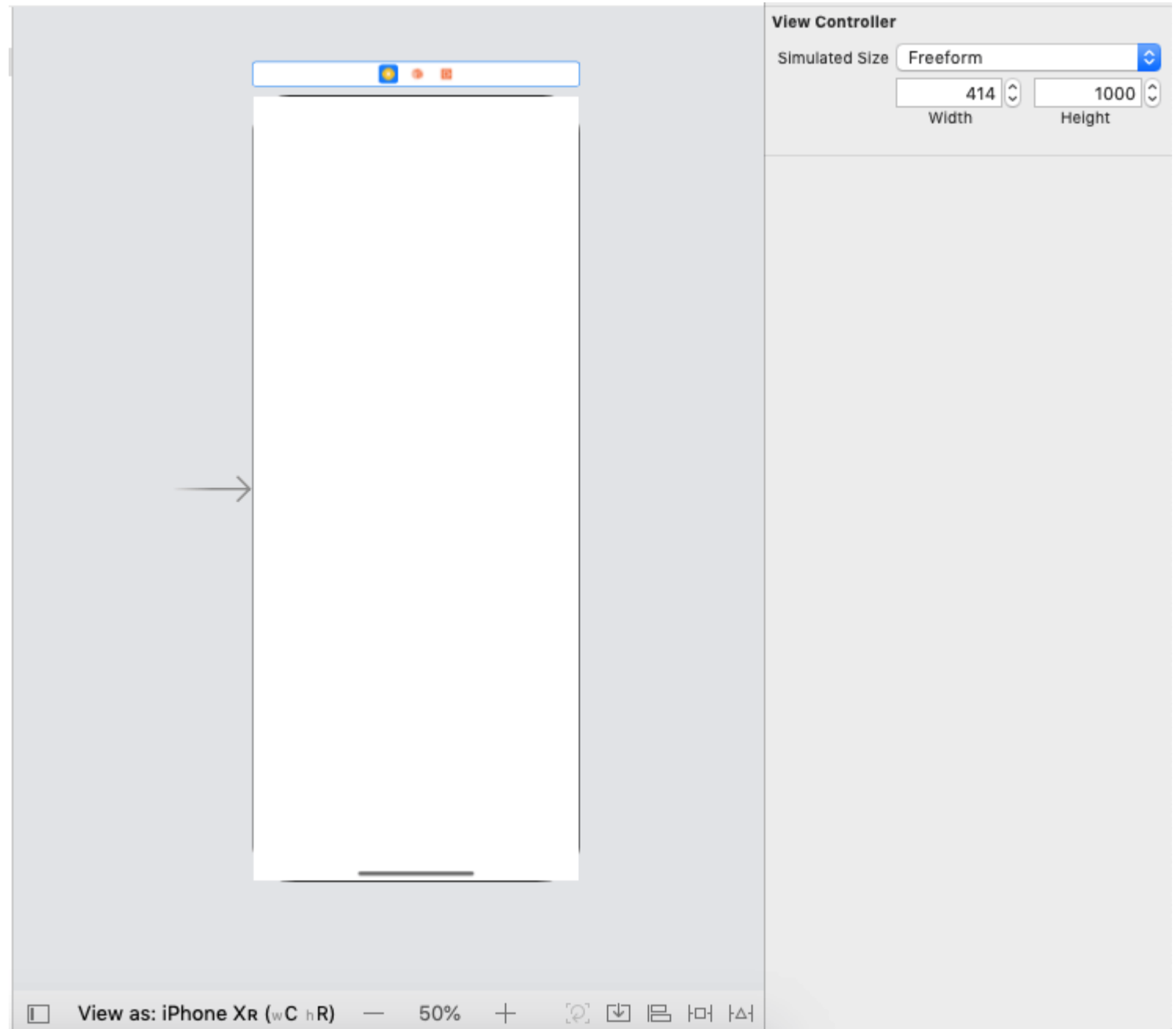
Otorgue el margen de restricción 0 al contentView desde ScrollView. También tenemos que igualar la altura y el ancho de ellos.



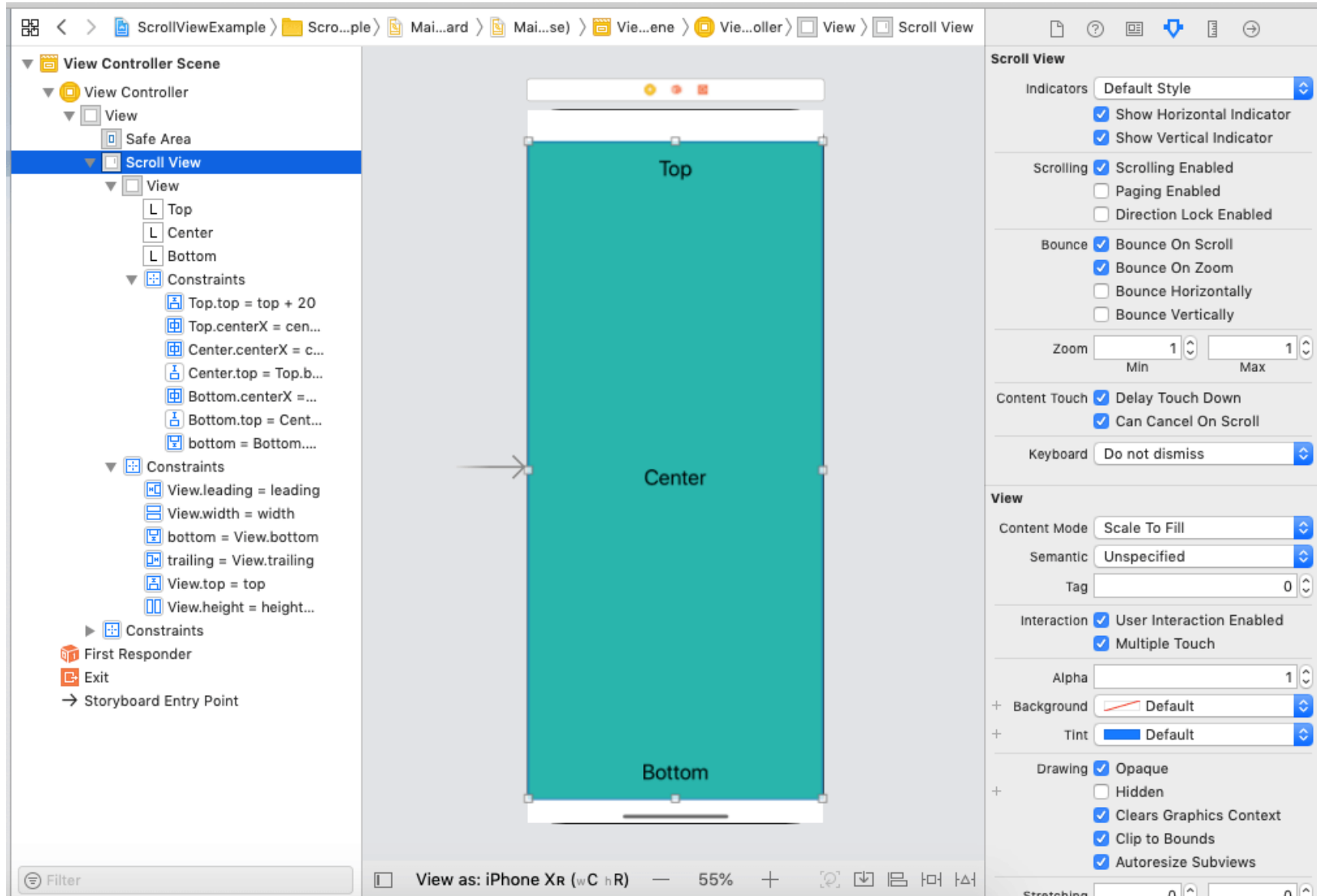
Establezca la prioridad de ContentView en baja, como se muestra en la imagen a continuación. Este es el paso más importante en esta configuración; de lo contrario, la vista de desplazamiento no se desplaza.



Tendremos el ScrollView y el ContentView en el guión gráfico. Dado que ScrollView funciona cuando el tamaño del contenido no cabe en toda la pantalla de la pantalla del iPhone. Definimos el tamaño de la pantalla ViewController en el inspector de tamaño en XCode. Por defecto, el tamaño simulado de ViewController es fijo. Sin embargo, tenemos que hacer que sea de forma libre y darle a la altura algo mayor que la altura actual de la pantalla como se muestra en la imagen a continuación.



Agregue un UILabel en la parte superior, central e inferior de ScrollView y proporcione el espacio vertical entre ellos para probar si ScrollView funciona o no. Después de agregar UILabels y cambiar el color de fondo, el generador de interfaz se verá como la siguiente ventana.



La configuración anterior hará que ScrollView to Scroll muestre todo el contenido en la pantalla.



Propiedades de UIScrollView

La clase UIScrollView contiene las siguientes propiedades.

SN	Propiedad	Tipo	Descripción
1	Delegate	UIScrollViewDelegate	Es el delegado del objeto UIScrollView.
2	contentSize	CGSize	Representa el tamaño de la vista de contenido.
3	contentOffset	CGPoint	Representa el punto en el que el origen de la vista de contenido está desplazado del origen de UIScrollView.
4 4	adjustedContentOffset	UIEdgeInsets	Representa las inserciones del contenido y el área segura de la vista de desplazamiento.
5 5	frameLayoutGuide	UILayoutGuide	La guía de diseño basada en el rectángulo de marco no transformado de la vista de desplazamiento.
6 6	contentLayoutGuide	UILayoutGuide	La guía de diseño basada en el rectángulo de contenido no traducido de la vista de desplazamiento.
7 7	isScrollEnabled	Bool	Representa el valor booleano que indica si la vista de desplazamiento está habilitada o no.
8	isDirectionLockEnabled	Bool	Representa el valor booleano que indica si la vista de desplazamiento se puede desplazar en una dirección particular.
9 9	isPagingEnabled	Bool	Es un valor booleano que representa si la paginación está habilitada en una dirección particular.
10	scrollsToTop	Bool	Es un valor booleano que controla si el desplazamiento al gesto superior está habilitado o no.
11	bounces	Bool	Es un valor booleano que representa si la vista de desplazamiento rebota sobre el borde del contenido.
12	alwaysBounceVertical	Bool	Es un valor booleano que representa si el rebote siempre ocurre cuando el desplazamiento vertical llega al final del contenido.
13	alwaysBounceHorizontal	Bool	Es un valor booleano que representa si el rebote siempre ocurre cuando el desplazamiento horizontal llega al final del contenido.
14	isTracking	Bool	Es un valor booleano que representa si el usuario toca el contenido para iniciar el desplazamiento.
15	isDragging	Bool	Es un valor booleano que representa si el usuario ha comenzado a desplazarse por el contenido.
dieciséis	isDecelerating	Bool	Es un valor booleano si el contenido se movía en la vista de desplazamiento cuando el usuario dejó de arrastrar o levantó el dedo.

17	desaccelerationRate	UIScrollView.DelcelerationRate	Es un valor de coma flotante que determina la tasa de desaceleración después de que el usuario levanta el dedo.
18 años	IndicationStyle	UIScrollView.IndicatorStyle	Representa el estilo de los indicadores de desplazamiento.
19	showsHorizontalScrollsIndicator	Bool	Es un valor booleano que representa si el indicador de desplazamiento horizontal es visible o no.
20	showsVerticalScrollIndicator	Bool	Es un valor booleano que representa si el indicador de desplazamiento vertical es visible o no.
21	refreshControl	UIRefreshControl	Representa el control de actualización asociado con la vista de desplazamiento.
22	canCancelContentTouches	Bool	Es un valor booleano que controla.
23	delayContentTouches	Bool	Es un valor booleano que representa si la vista de desplazamiento retrasa el manejo de los gestos de contacto.
24	directionPressGestureRecognizer	UIGestureRecognizer	El reconocedor de gestos subyacente para presionar botones direccionales.
25	panGestureRecognizer	UIPanGestureRecognizer	El reconocedor de gestos subyacente para gestos panorámicos.
26	pinchGestureRecognizer	UIPinchGestureRecognizer	El reconocedor de gestos subyacente para gestos de pellizco.
27	zoomScale	CGFloat	Es un valor de punto flotante que escala el zoom del contenido de la vista de desplazamiento.
28	maximumZoomScale	CGFloat	Es un valor máximo de coma flotante que se puede aplicar al zoom del contenido de la vista de desplazamiento.
29	minimumZoomScale	CGFloat	Es el valor mínimo de coma flotante que se puede aplicar al zoom del contenido de la vista de desplazamiento.
30	isZoomBouncing	Bool	Es un valor booleano que representa si el zoom ha excedido los límites de escala asociados.
31	isZooming	Bool	Es un valor booleano que representa si la vista de contenido está haciendo zoom actualmente o no.
32	bouncesZoom	Bool	Es un valor booleano que representa si la vista de desplazamiento anima la escala del contenido cuando la escala excede los límites máximos o mínimos.

Ejemplo

Este es un ejemplo simple en el que crearemos un ScrollView y ContentView. Crearemos dos UIViews dentro de la vista de desplazamiento para mostrar el contenido real.

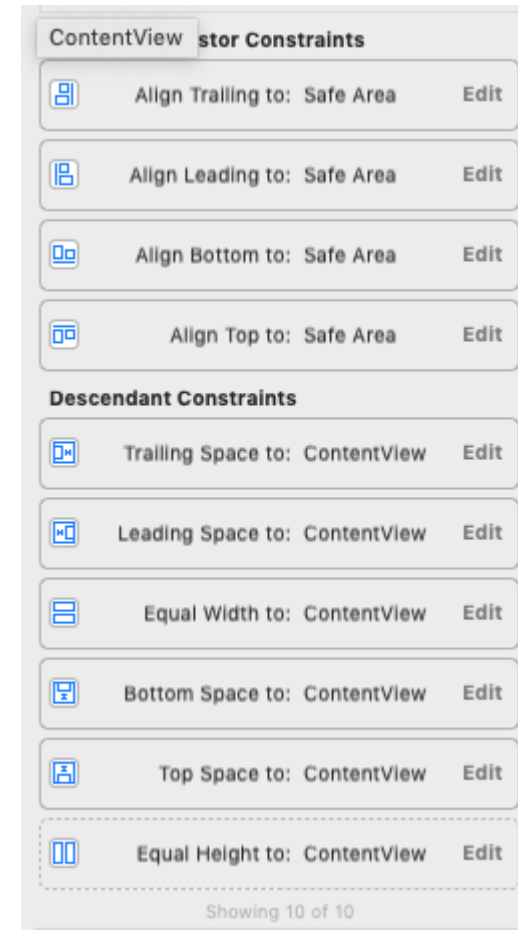
Interface Builder

En este ejemplo, utilizaremos la vista de desplazamiento, que contiene una vista de contenido. La vista de contenido se desplazará para mostrar dos uiviews. Para este propósito, use las instrucciones dadas en este tutorial para agregar la vista de desplazamiento al generador de interfaces y agregar la vista de desplazamiento a la interfaz.
















Agregue UIView a ScrollView y defina las reglas de diseño automático para UIView. Después de agregar el UIView (ContentView) a la vista de desplazamiento, necesitamos agregar dos UIViews y un UIButton al guión gráfico.

Para hacer que ScrollView sea desplazable, necesitamos cambiar el tamaño de ViewController de fijo a libre y definir un espacio vertical entre las vistas. Configure las reglas de diseño automático para ScrollView, ContentView, UIViews y el botón Enviar.

Reglas de diseño automático para ScrollView



Reglas de AutoLayout para ContentView

Sibling & Ancestor Constraints		
	Trailing Space to: Superview	Edit
	Leading Space to: Superview	Edit
	Equal Width to: Superview	Edit
	Bottom Space to: Superview	Edit
	Top Space to: Superview	Edit
	Equal Height to: Superview	Edit
Descendant Constraints		
	Align Center X to: Submit Btn	Edit
	Trailing Space to: User Detail... Equals: 10	Edit
	Trailing Space to: Admin Detail... Equals: 10	Edit
	Leading Space to: User Detail... Equals: 10	Edit
	Leading Space to: User Details Equals: 10	Edit
	Leading Space to: Admin Detail... Equals: 10	Edit
	Leading Space to: Admin Details Equals: 10	Edit
	Bottom Space to: Submit Btn Equals: 50	Edit
	Top Space to: Admin Details Equals: 54	Edit

Reglas de AutoLayout para AdminDetailsView

Sibling & Ancestor Constraints

Trailing Space to: Superview
Equals: 10

Edit

Leading Space to: Superview
Equals: 10

Edit

Height Equals: 220

Edit

Top Space to: Admin Details
Equals: 50

Edit

Bottom Space to: User Details
Equals: 30

Edit

Descendant Constraints

Trailing Space to: Address : G-...
Equals: 20

Edit

Trailing Space to: Contact : 99...
Equals: 20

Edit

Trailing Space to: Company Na...
Equals: 20

Edit

Trailing Space to: Admin Name...
Equals: 20

Edit

Leading Space to: Address : G-...
Equals: 20

Edit

Leading Space to: Contact : 99...
Equals: 20

Edit

Leading Space to: Company Na...
Equals: 20

Edit

Leading Space to: Admin Name...
Equals: 20















Edit

Top Space to: Admin Name...
Equals: 20

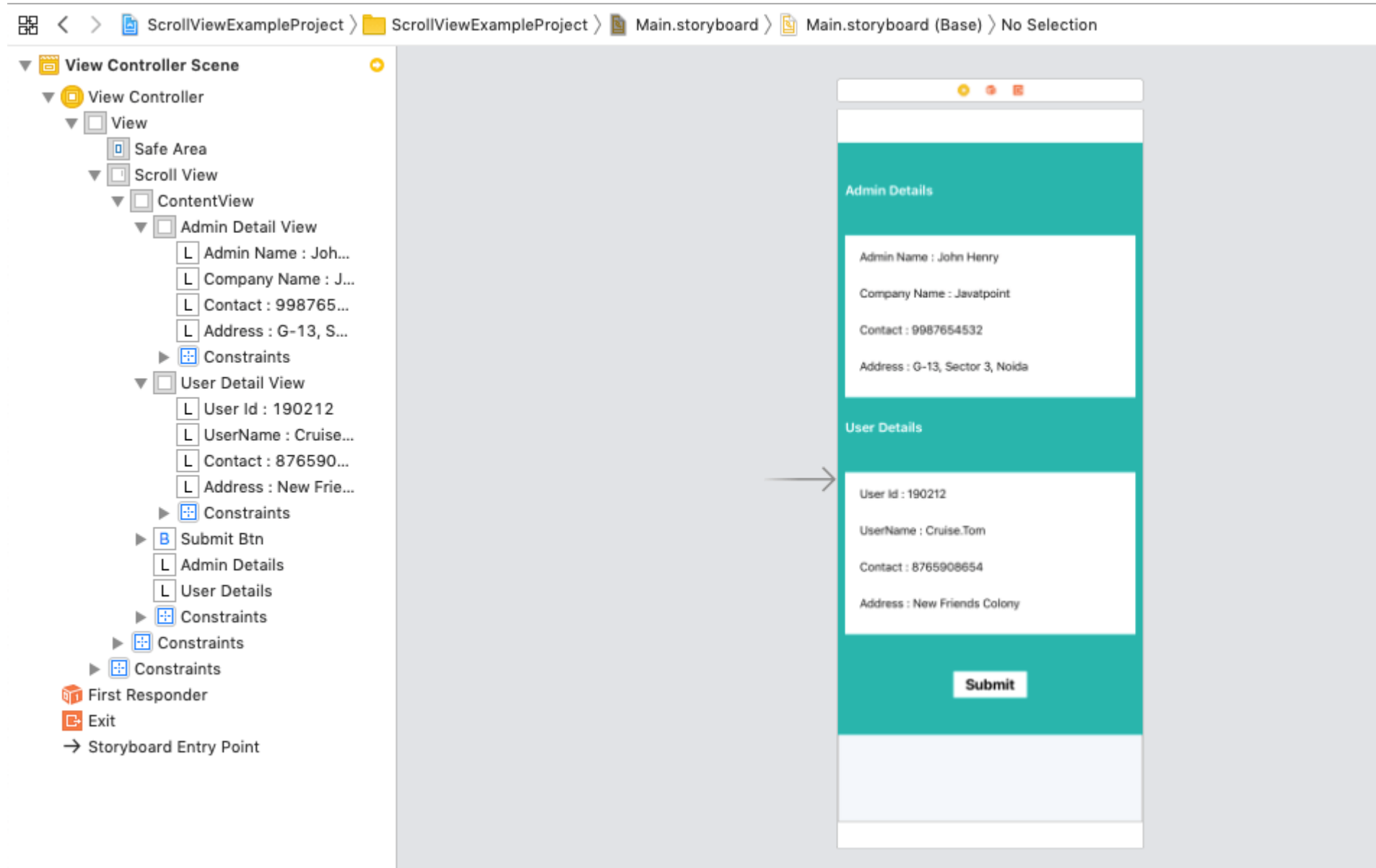
Edit

Showing 14 of 14

Reglas de diseño automático para UserDetailsView

Sibling & Ancestor Constraints		
	Trailing Space to: <i>Superview</i> Equals: 10	Edit
	Leading Space to: <i>Superview</i> Equals: 10	Edit
	Height Equals: 220	Edit
	Bottom Space to: Submit Btn Equals: 50	Edit
	Top Space to: User Details Equals: 50	Edit
Descendant Constraints		
	Trailing Space to: Address : Ne... Equals: 20	Edit
	Trailing Space to: Contact : 87... Equals: 20	Edit
	Trailing Space to: UserName :... Equals: 20	Edit
	Trailing Space to: User Id : 190... Equals: 20	Edit
	Leading Space to: Address : Ne... Equals: 20	Edit
	Leading Space to: Contact : 87... Equals: 20	Edit
	Leading Space to: UserName :... Equals: 20	Edit
	Leading Space to: User Id : 190... Equals: 20	Edit
	Top Space to: User Id : 190... Equals: 20	Edit
Showing 14 of 14		

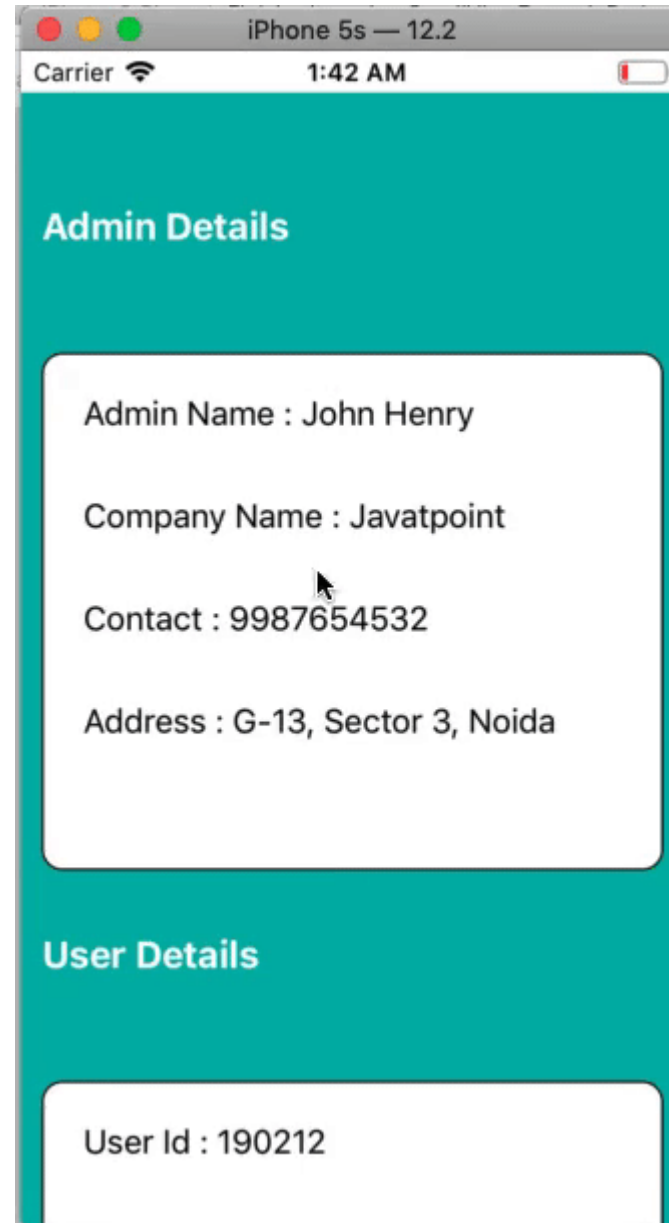
Main.storyboard



ViewController.swift

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    @IBOutlet weak var adminDetailView: UIView!
6.
7.    @IBOutlet weak var userDetailsView: UIView!
8.
9.    @IBOutlet weak var submitBtn: UIButton!
10.
11.    override func viewDidLoad() {
12.        super.viewDidLoad()
13.        // Do any additional setup after loading the view.
14.        adminDetailView.layer.cornerRadius = 10
15.        adminDetailView.layer.borderColor = UIColor.black.cgColor
16.        adminDetailView.layer.borderWidth = 1
17.        adminDetailView.layer.shadowOffset = CGSize(width: 2, height: 2)
18.
19.        adminDetailView.layer.shadowColor = UIColor.black.cgColor
20.        adminDetailView.layer.shadowRadius = 5
21.
22.        userDetailsView.layer.cornerRadius = 10
23.        userDetailsView.layer.borderColor = UIColor.black.cgColor
24.        userDetailsView.layer.borderWidth = 1
25.        userDetailsView.layer.shadowOffset = CGSize(width: 2, height: 2)
26.        userDetailsView.layer.shadowColor = UIColor.black.cgColor
27.        userDetailsView.layer.shadowRadius = 5
28.
29.        submitBtn.layer.cornerRadius = 10
30.        submitBtn.layer.borderColor = UIColor.black.cgColor
31.        submitBtn.layer.shadowColor = UIColor.black.cgColor
32.    }
33.
34.}
```

Salida:



iOS Content Views

Vista Indicador de actividad

ActivityIndicator es un tipo de vista de contenido que se puede usar para mostrar que una tarea está en progreso. Se utiliza principalmente en llamadas dentro de la red. Podemos controlar la Vista ActivityIndicator utilizando métodos definidos en la clase `UIActivityIndicatorView`. `UIActivityIndicatorView` es la subclase de `UIView`.

clase `UIActivityIndicatorView: UIView`

Podemos comenzar la animación llamando al método `startAnimating ()` de la clase `UIActivityIndicatorView`. Del mismo modo, el método `stopAnimating ()` se puede utilizar para detener la animación. Los dos métodos se usan en combinación para mostrar y ocultar el indicador de actividad en la aplicación iOS.

Agregar UIActivityIndicatorView a la interfaz

1. Busque UIActivityIndicatorView en la biblioteca de objetos y arrastre el resultado al guión gráfico.
2. Defina las reglas de diseño automático para gobernar el tamaño y la posición del indicador de actividad en los dispositivos de diferentes tamaños.
3. Cree una salida de acción de ActivityIndicator en la clase ViewController.
4. Utilice los métodos y propiedades definidos en la clase UIActivityIndicatorView para iniciar y detener la animación en consecuencia.

Métodos y propiedades.

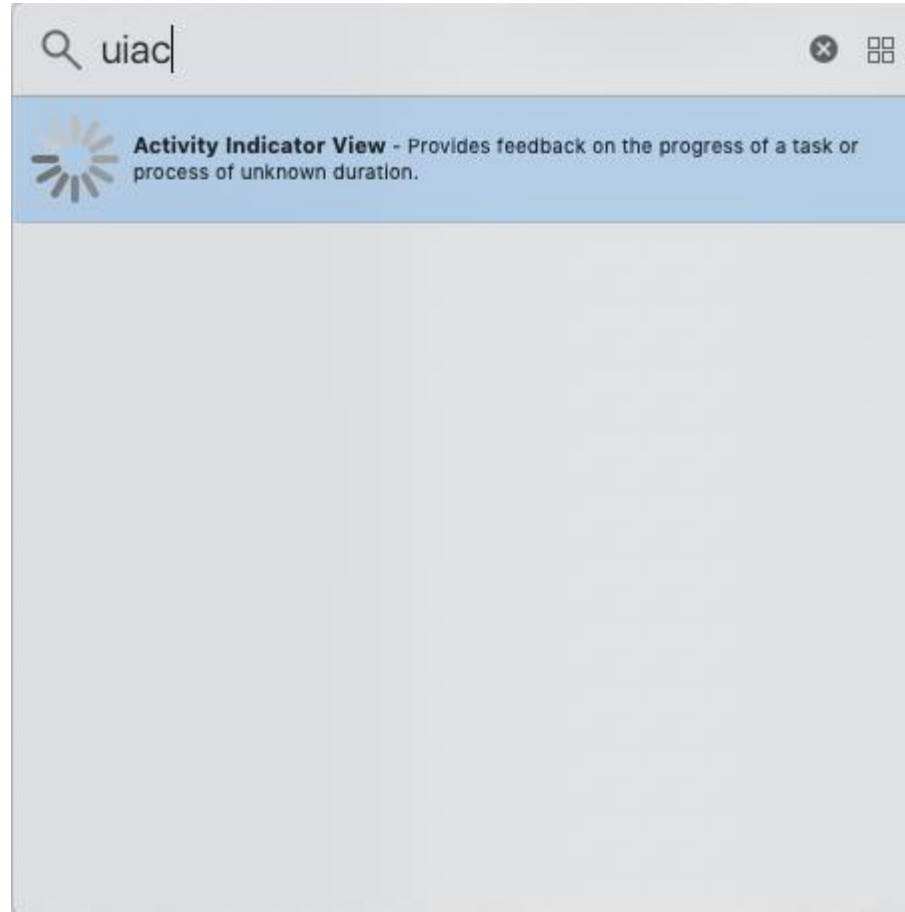
SN	Método o propiedad	Descripción
1	func startAnimating ()	Este método se utiliza para iniciar la animación del indicador de actividad.
2	func stopAnimating ()	Este método se utiliza para detener la animación del indicador de actividad.
3	var isAnimating: Bool	Esta es una propiedad booleana que indica si el indicador de actividad está animando o no.
4 4	var hidesWhenStopped (): Bool	Esta es una propiedad booleana que controla si el receptor está oculto cuando se detiene la animación.
5 5	Var style: UIActivityIndicatorView.style	Es la apariencia básica del indicador de actividad.
6 6	Var color: UIColor	Determina el color del indicador de actividad.

Ejemplo 1

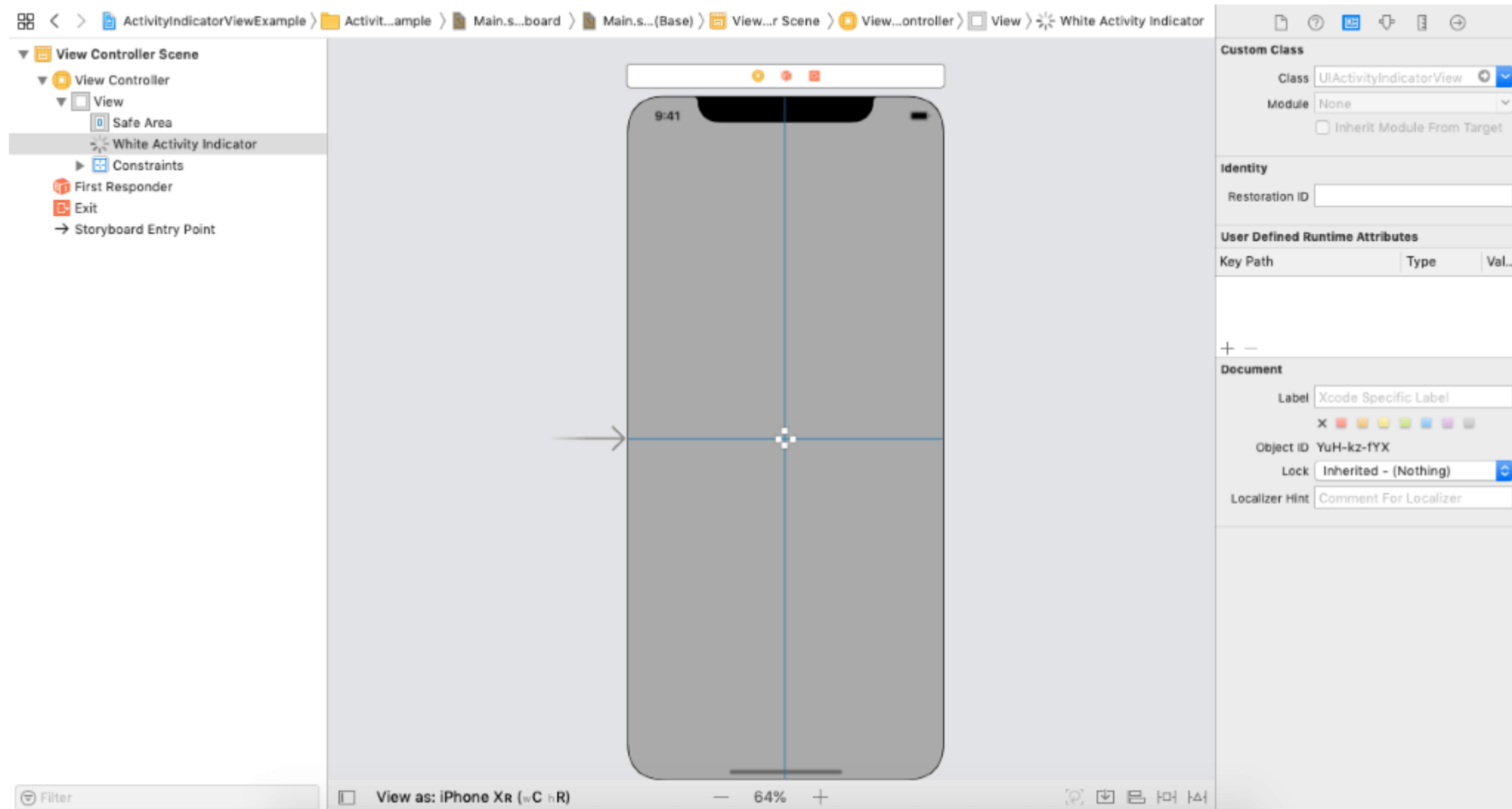
Este es un ejemplo simple que visualiza la vista del indicador de actividad en la pantalla.

Interface Builder

En este ejemplo, primero agregaremos `ActivityIndicatorView` al guión gráfico. Para este propósito, buscaremos el `UIActivityIndicatorView` en la biblioteca de objetos y arrastraremos el resultado al guión gráfico.



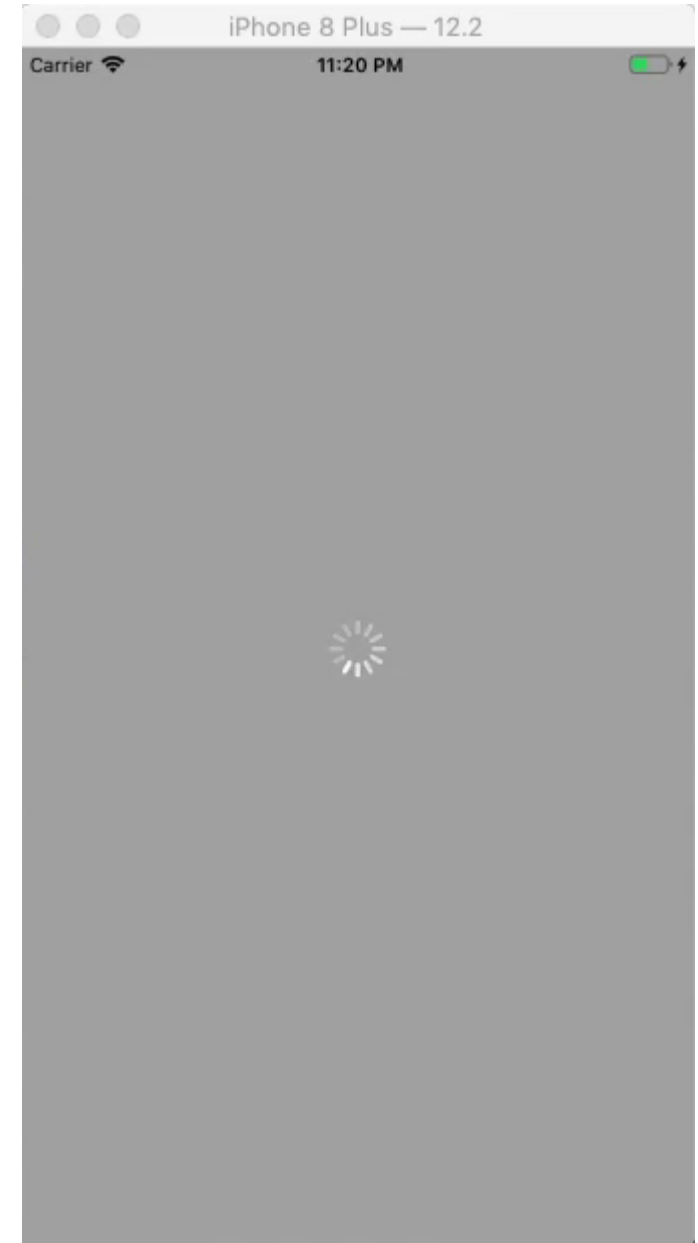
Defina las reglas de diseño automático para el indicador de actividad y cree la salida de conexión en el controlador de vista.



ViewController.swift

```
1.import UIKit
2.
3.class ViewController: UIViewController {
4.
5.    @IBOutlet weak var activityIndicator: UIActivityIndicatorView!
6.    override func viewDidLoad() {
7.        super.viewDidLoad()
8.        // Do any additional setup after loading the view.
9.        activityIndicator.style = .whiteLarge
10.        activityIndicator.startAnimating()
11.    }
12.
13.}
```

Output:

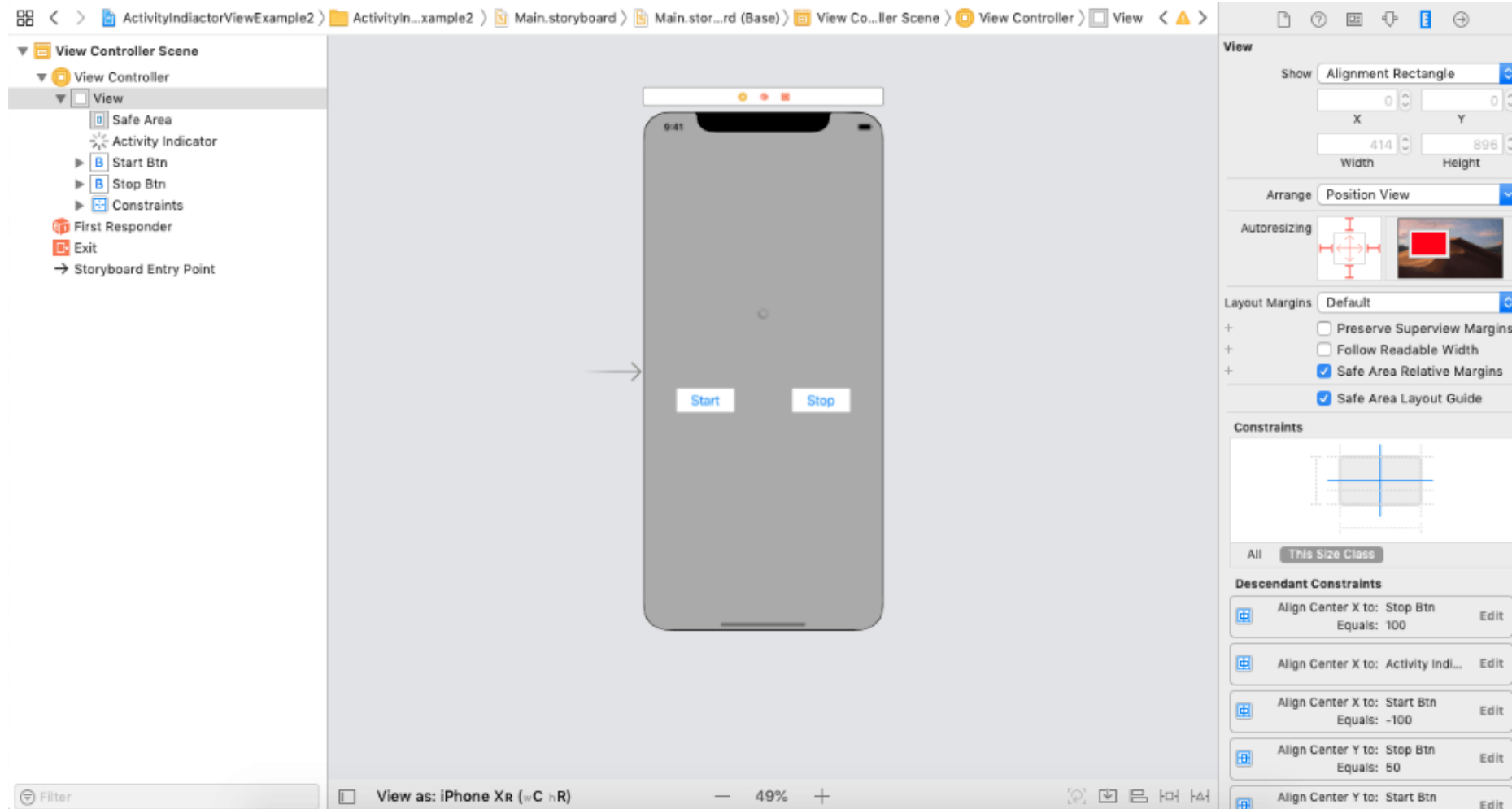


Ejemplo 2

En el ejemplo 1, hemos mostrado un UIActivityIndicatorView no controlado. En este ejemplo, crearemos dos botones para iniciar y detener la animación del indicador de actividad. En la salida de acción del botón de inicio, comenzaremos la animación, mientras que, en la salida de acción del botón de parada, detendremos la animación.

Interface Builder

La siguiente imagen muestra el guión gráfico que crearemos en el ejemplo. Esto es similar al ejemplo 1 en la mayoría de los casos. En este ejemplo, también agregaremos los botones para controlar el comportamiento de la vista del indicador de Actividad.



ViewController.swift

```
1.import UIKit
2.
3.
4.class ViewController: UIViewController {
5.
6.
7.    @IBOutlet weak var startBtn: UIButton!
8.    @IBOutlet weak var activityIndicator: UIActivityIndicatorView!
9.    @IBOutlet weak var stopBtn: UIButton!
10.    override func viewDidLoad() {
11.        super.viewDidLoad()
12.        // Do any additional setup after loading the view.
13.        activityIndicator.style = .whiteLarge
14.        activityIndicator.hidesWhenStopped = true
15.        startBtn.layer.cornerRadius = 10
16.        startBtn.layer.borderColor = UIColor.black.cgColor
17.        startBtn.layer.borderWidth = 1
18.        stopBtn.layer.cornerRadius = 10
19.        stopBtn.layer.borderWidth = 1
20.        stopBtn.layer.borderColor = UIColor.black.cgColor
21.    }
22.
23.
24.    @IBAction func clickedStartBtn(_ sender: Any) {
25.        activityIndicator.startAnimating()
26.    }
27.
28.    @IBAction func clickedStopBtn(_ sender: Any) {
29.        activityIndicator.stopAnimating()
30.    }
31.
32.}
```

Output:

