TECNOLOGICO NACIONAL DE MEXICO

Campus La Laguna

Ingeniería en Sistemas Computacionales

DESARROLLO EN IOS

SEMESTRE: Ago – Dic / 2022

Documentación

ALUMNO:

19130519 Roberto Esquivel Troncoso

PROFESOR:

VALDES ALVARADO MARTIN OSWALDO

Torreón, Coah. A 16 de diciembre de 2022

Índice

¿Qué es el modelo MVVM? 2
appPeliculasSwift 3
appSwiftTabBar 11
appDireccion 40
Referencias: 42

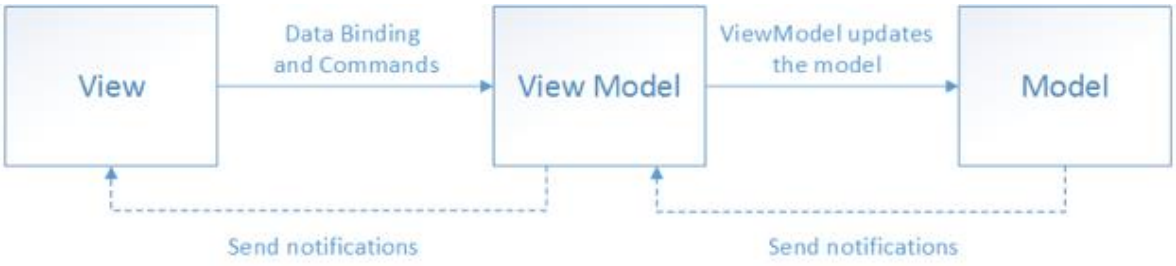
¿Qué es el modelo MVVM?

La Xamarin.Forms experiencia del desarrollador normalmente implica crear una interfaz de usuario en XAML y, a continuación, agregar código subyacente que funciona en la interfaz de usuario. A medida que se modifican las aplicaciones y aumenta el tamaño y el ámbito, pueden surgir problemas de mantenimiento complejos. Estos problemas incluyen el acoplamiento estricto entre los controles de interfaz de usuario y la lógica de negocios, lo que aumenta el costo de realizar modificaciones de la interfaz de usuario y la dificultad de probar este código unitaria.

El patrón Model-View-ViewModel (MVVM) ayuda a separar limpiamente la lógica de negocios y presentación de una aplicación de su interfaz de usuario (UI). Mantener una separación limpia entre la lógica de la aplicación y la interfaz de usuario ayuda a abordar numerosos problemas de desarrollo y puede facilitar la prueba, el mantenimiento y la evolución de una aplicación. También puede mejorar considerablemente las oportunidades de reutilización del código y permite a los desarrolladores y diseñadores de interfaz de usuario colaborar más fácilmente al desarrollar sus respectivas partes de una aplicación.

El patrón MVVM

Hay tres componentes principales en el patrón MVVM: el modelo, la vista y el modelo de vista. Cada uno sirve para un propósito distinto. En la siguiente figura se muestran las relaciones entre los tres componentes.



Además de comprender las responsabilidades de cada componente, también es importante comprender cómo interactúan entre sí. En un nivel alto, la vista "conoce" el modelo de vista y el modelo de vista "conoce" el modelo, pero el modelo no es consciente del modelo de vista y el modelo de vista no es consciente de la vista.

Las ventajas de usar el patrón MVVM son las siguientes:

- Si hay una implementación de modelo existente que encapsula la lógica de negocios existente, puede ser difícil o arriesgada cambiarla. En este escenario, el modelo de vista actúa como adaptador para las clases de modelo y le permite evitar realizar cambios importantes en el código del modelo.
- Los desarrolladores pueden crear pruebas unitarias para el modelo de vista y el modelo, sin usar la vista. Las pruebas unitarias del modelo de vista pueden ejercer exactamente la misma funcionalidad que la vista.

- La interfaz de usuario de la aplicación se puede rediseñar sin tocar el código, siempre que la vista se implemente completamente en XAML. Por lo tanto, una nueva versión de la vista debe funcionar con el modelo de vista existente.
- Los diseñadores y desarrolladores pueden trabajar de forma independiente y simultánea en sus componentes durante el proceso de desarrollo. Los diseñadores pueden centrarse en la vista, mientras que los desarrolladores pueden trabajar en el modelo de vista y los componentes del modelo.

appPeliculasSwift

ContentView.swift

```
import SwiftUI

struct ContentView: View {
   // Obtenemos los modelos de las bases de datos
   @StateObject var viewModel = MoviesViewModel() //MovieViewModel.swift
   // ponemos una bandera en falso la seleccion de la pelicula
   @State var presentAddMovieSheet = false
   // Creamos un boton, para agregar peliculas
   private var addButton: some View {
     Button(action: { // y cuando se seleccione quitamos la bandera
         self.presentAddMovieSheet.toggle() }) {
       Image(systemName: "plus")
     }
   }
   // creamos una pila, para agregar una pelicula
   private func movieRowView(movie: Movie) -> some View {
      NavigationLink(destination: MovieDetailsView(movie: movie)) { // obtenemos las
peliculas
        VStack(alignment: .leading) {
          Text(movie.title) // titulo de la peliculas
            .font(.headline)
          Text(movie.year) // año de la pelicula
            .font(.subheadline)
        }
      }
   }
   var body: some View {
     NavigationView {
       List {
          // ver las peliculas
         ForEach (viewModel.movies) { movie in
           movieRowView(movie: movie)
         } // boton para eliminar
         .onDelete() { indexSet in
           // se elimina la pelicula con el index
           viewModel.removeMovies(atOffsets: indexSet)
```

```
          }
        }.blendMode(/*@START_MENU_TOKEN@*/.darken/*@END_MENU_TOKEN@*/)
        .navigationBarTitle("Peliculas") // titulo de la navegacion
        .navigationBarItems(trailing: addButton) // boton para agregar
        .onAppear() {
          print("Hola.")
          self.viewModel.subscribe()
        }
        .sheet(isPresented: self.$presentAddMovieSheet) {
          MovieEditView() //MovieEditView.swift
        }
         // color de texto
      }.foregroundColor(Color.black)
           .background(Color.blue)
        // End Navigation
    }
    // End Body
}
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

appPeliculasSwiftApp.swift

```
import SwiftUI
import Firebase
@main
struct appPeliculasSwiftApp: App {
    init() {
        FirebaseApp.configure() // inicio de la configuracion de la base de datos
        }

        var body: some Scene {
            WindowGroup {
                ContentView()
            }
        }
}
```

Models/Movie.swift

```
import Foundation
import FirebaseFirestoreSwift

struct Movie: Identifiable, Codable { // datos de la pelicula
  @DocumentID var id: String?
  var title: String
```

```
  var description: String
  var year: String

  enum CodingKeys: String, CodingKey { // llaves de la pelicula
    case id
    case title
    case description
    case year
  }
}
```

View/MovieDetailsView.swift

```
import SwiftUI

struct MovieDetailsView: View {
    @Environment(\.presentationMode) var presentationMode
    @State var presentEditMovieSheet = false // bandera para el boton editar

    var movie: Movie

    private func editButton(action: @escaping () -> Void) -> some View { // Boton editar
      Button(action: { action() }) {
        Text("Editar")
      }
    }

    var body: some View {
      Form {
        Section(header: Text("Pelicula")) { // vista para la pelicula seleccionada
          Text(movie.title) // Titulo y descripcion
          Text(movie.description)

        }

        Section(header: Text("Año")) { // año de la pelicula
            Text(movie.year)
        }
      }
      .navigationBarTitle(movie.title)
      .navigationBarItems(trailing: editButton { // boton editar
        self.presentEditMovieSheet.toggle() // cambia la bandera del boton
      })
      .onAppear() {
        print("MovieDetailsView.onAppear() for \(self.movie.title)")
      }
```

```
      .onDisappear() {
        print("MovieDetailsView.onDisappear()")
      }
      .sheet(isPresented: self.$presentEditMovieSheet) {
        MovieEditView(viewModel: MovieViewModel(movie: movie), mode: .edit) { result in
          if case .success(let action) = result, action == .delete {
            self.presentationMode.wrappedValue.dismiss()
          }
        }
      }
    }

  }

struct MovieDetailsView_Previews: PreviewProvider {
    static var previews: some View {
        let movie = Movie(title: "titulo pelicula", description: "este es un ejemplo de
descripcion", year: "2021")
        return
          NavigationView {
            MovieDetailsView(movie: movie)
          }
    }
}
```

View/MovieEditView.swift

```
import SwiftUI

// modos para crear y editar
enum Mode {
  case new
  case edit
}

// modos de accion para eliminar, aceptar y cancelars
enum Action {
  case delete
  case done
  case cancel
}


struct MovieEditView: View {
    @Environment(\.presentationMode) private var presentationMode
    @State var presentActionSheet = false
```

```
@ObservedObject var viewModel = MovieViewModel()
var mode: Mode = .new
var completionHandler: ((Result<Action, Error>) -> Void)?


var cancelButton: some View {
  Button(action: { self.handleCancelTapped() }) {
    Text("Cancelar")
  }
}

var saveButton: some View {
  Button(action: { self.handleDoneTapped() }) {
    Text(mode == .new ? "Done" : "Guardar")
  }
  .disabled(!viewModel.modified)
}

var body: some View {
  NavigationView {
    Form {
      Section(header: Text("Pelicula")) {
        TextField("Titulo", text: $viewModel.movie.title)
        TextField("Año", text: $viewModel.movie.year)
      }

      Section(header: Text("Descripcion")) {
        TextField("Descripcion", text: $viewModel.movie.description)
      }

      if mode == .edit {
        Section {
          Button("Eliminar Pelicula") { self.presentActionSheet.toggle() }
            .foregroundColor(.red)
        }
      }
    }
    .navigationTitle(mode == .new ? "Nueva Pelicula" : viewModel.movie.title)
    .navigationBarTitleDisplayMode(mode == .new ? .inline : .large)
    .navigationBarItems(
      leading: cancelButton,
      trailing: saveButton
    )
    .actionSheet(isPresented: $presentActionSheet) {
      ActionSheet(title: Text("Estas seguro?"),
                  buttons: [
                    .destructive(Text("Eliminar Pelicula"),
```

```
                        action: { self.handleDeleteTapped() }),
                .cancel()
              ])
        }
      }
    }

    // Action Handlers

    func handleCancelTapped() {
      self.dismiss()
    }

    func handleDoneTapped() {
      self.viewModel.handleDoneTapped()
      self.dismiss()
    }

    func handleDeleteTapped() {
      viewModel.handleDeleteTapped()
      self.dismiss()
      self.completionHandler?(.success(.delete))
    }

    func dismiss() {
      self.presentationMode.wrappedValue.dismiss()
    }
  }

struct MovieEditView_Previews: PreviewProvider {
  static var previews: some View {
    let movie = Movie(title: "Titulo ejemplo", description: "Ejemlo descripcion", year: "2020")
    let movieViewModel = MovieViewModel(movie: movie)
    return MovieEditView(viewModel: movieViewModel, mode: .edit)
  }
}
```

ViewModels/ MoviesViewModel.swift

```
import Foundation
import Combine
import FirebaseFirestore

class MoviesViewModel: ObservableObject {
    @Published var movies = [Movie]()

     private var db = Firestore.firestore()
     private var listenerRegistration: ListenerRegistration?
```

```
    deinit {
      unsubscribe()
    }

   func unsubscribe() {
       if listenerRegistration != nil {
         listenerRegistration?.remove()
         listenerRegistration = nil
       }
     }
     // listar las peliculas
     func subscribe() {
       if listenerRegistration == nil {
         listenerRegistration = db.collection("movielist").addSnapshotListener {
(querySnapshot, error) in
           guard let documents = querySnapshot?.documents else {
             print("No documents")
             return
           }

           self.movies = documents.compactMap { queryDocumentSnapshot in
             try? queryDocumentSnapshot.data(as: Movie.self)
           }
         }
       }
     }
      // eliminar peliculass
     func removeMovies(atOffsets indexSet: IndexSet) {
       let movies = indexSet.lazy.map { self.movies[$0] }
       movies.forEach { movie in
         if let documentId = movie.id {
           db.collection("movielist").document(documentId).delete { error in
             if let error = error {
               print("Unable to remove document: \(error.localizedDescription)")
             }
           }
         }
       }
     }
}
```

ViewModels/ MovieViewModel.swift

```
import Foundation
import Combine
```

```
import FirebaseFirestore

class MovieViewModel: ObservableObject {

  @Published var movie: Movie
  @Published var modified = false

  private var cancellables = Set<AnyCancellable>()

  init(movie: Movie = Movie(title: "", description: "", year: "")) {
    self.movie = movie

    self.$movie
      .dropFirst()
      .sink { [weak self] movie in
        self?.modified = true
      }
      .store(in: &self.cancellables)
  }

  // Firestore

  private var db = Firestore.firestore()

  private func addMovie(_ movie: Movie) {
    do {
      let _ = try db.collection("movielist").addDocument(from: movie)
    }
    catch {
      print(error)
    }
  }

  private func updateMovie(_ movie: Movie) {
    if let documentId = movie.id {
      do {
        try db.collection("movielist").document(documentId).setData(from: movie)
      }
      catch {
        print(error)
      }
    }
  }

  private func updateOrAddMovie() {
    if let _ = movie.id {
      self.updateMovie(self.movie)
```

```
    }
    else {
      addMovie(movie)
    }
  }

  private func removeMovie() {
    if let documentId = movie.id {
      db.collection("movielist").document(documentId).delete { error in
        if let error = error {
          print(error.localizedDescription)
        }
      }
    }
  }

  // UI handlers

  func handleDoneTapped() {
    self.updateOrAddMovie()
  }

  func handleDeleteTapped() {
    self.removeMovie()
  }

}
```

appSwiftTabBar

CMatematicas.swift

```
import Foundation

class CMatematicas {
    // Suma
    static func Sumar (n1: Int, n2: Int) -> Int {
        let result =  n1 + n2
        return result
    }
    static func Sumar (n1: Double, n2: Double) -> Double {
        let result = n1 + n2
        return result
    }
    // Resta
    static func Resta (n1: Int, n2: Int) -> Int {
        return n1 - n2
```

```
    }
    static func Resta (n1: Double, n2: Double) -> Double {
        return n1 - n2
    }
    // Multi
    static func Multi (n1: Int, n2: Int) -> Int {
        return n1 * n2
    }
    static func Multi (n1: Double, n2: Double) -> Double {
        return n1 * n2
    }
    // Div
    static func Div (n1: Int, n2: Int) -> Int {
        return n1 / n2
    }
    static func Div (n1: Double, n2: Double) -> Double {
        return n1 / n2
    }
    // Modulo
    static func Mod (n1: Int, n2: Int) -> Int {
        return n1 % n2
    }
}
```

DegradadoLineal.swift

```
import UIKit

class DegradadoLineal: UIView {


    // Only override draw() if you perform custom drawing.
    // An empty implementation adversely affects performance during animation.
    override func draw(_ rect: CGRect) {
        // Drawing code
        let canvas = UIGraphicsGetCurrentContext()
        canvas?.setLineWidth(3.0)
        // Gradientes
        let posicion : [CGFloat] = [0.0,0.25,0.75]

        //let colores = [UIColor.darkGray.cgColor, UIColor.green.cgColor,
UIColor.blue.cgColor, UIColor.cyan.cgColor]

        let colores = [UIColor.brown.cgColor, UIColor.purple.cgColor,
UIColor.orange.cgColor, UIColor.yellow.cgColor]
        let colorSpace1 = CGColorSpaceCreateDeviceRGB()
```

```
        let gradiente = CGGradient(colorsSpace: colorSpace1, colors: colores as CFArray,
locations: posicion)

        var startPoint = CGPoint()
        var endPoint = CGPoint()
        startPoint.x = 0.0
        startPoint.y = 0.0
        endPoint.x = rect.width
        endPoint.y = rect.height

        canvas?.drawLinearGradient(gradiente!, start: startPoint, end: endPoint, options:
.drawsBeforeStartLocation)

        // Curvas de bezier
        canvas?.move(to: CGPoint(x: 0, y: 0))
        canvas?.addCurve(to: CGPoint(x: rect.width-10, y: 400), control1: CGPoint(x: 20, y:
200), control2: CGPoint(x: rect.width-50, y: 50))
        canvas?.strokePath()
    }
}
```

Graficos2D.swift

```
import UIKit
class Graficos2D: UIView {
    @IBOutlet weak var segmented: UISegmentedControl!

    @IBOutlet weak var txtX1: UITextField!
    @IBOutlet weak var txtY1: UITextField!
    @IBOutlet weak var txtX2: UITextField!
    @IBOutlet weak var txtY2: UITextField!

    var x1 : CGFloat = 200
    var y1 : CGFloat = 200
    var x2 : CGFloat = 400
    var y2 : CGFloat = 400

    @IBOutlet weak var SliderX1: UISlider!
    @IBOutlet weak var SliderX2: UISlider!
    @IBOutlet weak var SliderY1: UISlider!
    @IBOutlet weak var SliderY2: UISlider!

    var seleccionado : Int = 0


    // Only override draw() if you perform custom drawing.
    // An empty implementation adversely affects performance during animation.
    override func draw(_ rect: CGRect) {
```

```
        // Drawing code
        let canvas = UIGraphicsGetCurrentContext()
        canvas?.setLineWidth(1.0)

        // Gradientes
        let posicion : [CGFloat] = [0.0,0.25,0.75]

        //let colores = [UIColor.darkGray.cgColor, UIColor.green.cgColor,
UIColor.blue.cgColor, UIColor.cyan.cgColor]

        let colores = [UIColor.brown.cgColor, UIColor.purple.cgColor,
UIColor.orange.cgColor, UIColor.yellow.cgColor]
        let colorSpace1 = CGColorSpaceCreateDeviceRGB()
        let gradiente = CGGradient(colorsSpace: colorSpace1, colors: colores as CFArray,
locations: posicion)

        var startPoint = CGPoint()
        var endPoint = CGPoint()
        startPoint.x = 0
        startPoint.y = 0
        endPoint.x = rect.width
        endPoint.y = rect.height

        canvas?.drawLinearGradient(gradiente!, start: startPoint, end: endPoint, options:
.drawsBeforeStartLocation)

        canvas?.setStrokeColor(#colorLiteral(red: Float(drand48()),green: Float(drand48()),
blue: Float(drand48()), alpha: 1.0))

        if seleccionado == 0{
            // GRADIENT
            // Gradiente Radial
            let locationR : [CGFloat] = [0.0,0.5,1.0]
            let coloresR = [UIColor.brown.cgColor, UIColor.purple.cgColor,
UIColor.orange.cgColor, UIColor.yellow.cgColor]

            let colorSpaceRadial = CGColorSpaceCreateDeviceRGB()

            let gradienteRadial = CGGradient(colorsSpace: colorSpaceRadial, colors: coloresR
as CFArray, locations: locationR)

            var startPointR = CGPoint()
            var endPointR = CGPoint()
            startPointR.x = x1
            startPointR.y = y1
            endPointR.x = x2
            endPointR.y = y2
```

```
            // Se requieren dos radios
            let radio1 : CGFloat = 90.0
            let radio2 : CGFloat = -50.0
            canvas?.drawRadialGradient(gradienteRadial!, startCenter: startPointR,
startRadius: radio1, endCenter: endPointR, endRadius: radio2, options: [])
        } else if seleccionado == 1 {
            // Curvas de bezier
            canvas?.move(to: CGPoint(x: x1, y: y1))
            canvas?.addCurve(to: CGPoint(x: x2, y: y2),
                             control1: CGPoint(x: 20, y: 200),
                             control2: CGPoint(x: rect.width-50, y: 50))
            canvas?.strokePath()
        } else if seleccionado == 2 {
            var midx : CGFloat = 0.0
            var midy : CGFloat = 0.0

            midx = rect.width/2
            midy = rect.height/2
            var y = rect.height


            for x in stride(from: midx-150, through: midx, by: 10) { // x = midx-250; x <= midx;
x+=10
                y = midy - (x-60) * 0.4
                canvas?.move(to: CGPoint(x: x, y: midy))
                canvas?.addLine(to: CGPoint(x: midx, y: y))
                canvas?.strokePath()
            }

            for x in stride(from: midx, through: rect.height/2 - 89, by: 10) { //x = midx; x <= 570;
x+=10
                y = midy - 100 + (x-midx+100) * 0.4
                canvas?.move(to: CGPoint(x: x, y: midy))
                canvas?.addLine(to: CGPoint(x: midx, y: y))
                canvas?.strokePath()
            }

            for x in stride(from: midx+250, through: midx, by: -10) { //x = midx+250; x >= midx;
x-=10
                y = midy + 100 - (x-midx) * 0.4
                canvas?.move(to: CGPoint(x: x, y: midy))
                canvas?.addLine(to: CGPoint(x: midx, y: y))
                canvas?.strokePath()
            }

            for x in stride(from:  midx, through:-35, by: -10) { //x = midx; x >= 70; x-=10
```

```
            y = midy + 100 - (midx-x) * 0.4
            canvas?.move(to: CGPoint(x: x, y: midy))
            canvas?.addLine(to: CGPoint(x: midx, y: y))
            canvas?.strokePath()
        }
      }
    }

    @IBOutlet weak var btnDibujar: UIButton!

    @IBAction func btnDibujar(_ sender: UIButton) {
        x1 = CGFloat(Int(SliderX1.value))
        y1 = CGFloat(Int(SliderY1.value))
        x2 = CGFloat(Int(SliderX2.value))
        y2 = CGFloat(Int(SliderY2.value))

        seleccionado = self.segmented.selectedSegmentIndex

        self.setNeedsDisplay()

    }

    @IBAction func segmented(_ sender: UISegmentedControl) {

    }
}
```

Vista1VC.swift

```
import UIKit

class Vista1VC: UIViewController {


    var num : Int = 0
    var num2 : Int = 0
    var resultado:Int = 0
    var resuString = ""

    @IBOutlet weak var labelNumero: UILabel!
    @IBOutlet weak var slider1: UISlider!
    @IBOutlet weak var stepper1: UIStepper!
    @IBOutlet weak var segmento: UISegmentedControl!
    @IBOutlet weak var texto1: UITextField!

    @IBOutlet weak var txtOperacion: UILabel!
```

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    labelNumero.text = String(slider1.value);
}

@IBAction func slider1Action(_ sender: UISlider) {
    labelNumero.text = "\(String(describing: lround(Double(sender.value))))"
    stepper1.value = Double(sender.value)
    self.segmentedAction(segmento)
}

@IBAction func stepper1Action(_ sender: UIStepper) {

    labelNumero.text = "\(String(describing: lround(Double(sender.value))))"
    slider1.value = Float(Double(sender.value))
    self.segmentedAction(segmento)
}
@IBAction func segmentedAction(_ sender: UISegmentedControl) {
    let indice : Int = sender.selectedSegmentIndex
    let num = Int(labelNumero.text!)

    if indice == 0 {

        if num! >= 0 {
            let n = String(num!, radix: 2)
            texto1.text = n
        }else if num! < 0 {
            let n = 256 - (-1 * num!)
            texto1.text = String(n, radix: 2)
        }
    }
    else if indice == 1 {
        if num! >= 0 {
            let n = String(num!, radix: 8)
            texto1.text = n
        } else if num! < 0 {
            let n = 256 - (-1 * num!)
            texto1.text = String(n, radix: 8)
        }

    }
    else if indice == 2 {
        let n = String(num!, radix: 16).uppercased()
        texto1.text = n
    }
}
```

```
@IBAction func btnSuma(_ sender: UIButton) {
    //num = Int(String(labelNumero.text!))!
    if num == 0 {
        num = Int(String(labelNumero.text!))!
        resuString += String(num)
    }
    else{
        num2 = Int(String(labelNumero.text!))!
        resultado += CMatematicas.Sumar(n1: num, n2: num2)

        resuString += " + " + String(num2)
    }
    txtOperacion.text = resuString
}


@IBAction func btnResta(_ sender: UIButton) {
    if num == 0 {
        num = Int(String(labelNumero.text!))!

        resuString += String(num)
    }
    else{
        num2 = Int(String(labelNumero.text!))!
        resultado += CMatematicas.Resta(n1: num, n2: num2)
        resuString += " - " + String(num2)
    }
    txtOperacion.text = resuString
}

@IBAction func btnDiv(_ sender: UIButton) {
    if num == 0 {
        num = Int(String(labelNumero.text!))!

        resuString += String(num)
    }
    else{
        num2 = Int(String(labelNumero.text!))!
        resultado += CMatematicas.Div(n1: num, n2: num2)

        resuString += " / " + String(num2)
    }
    txtOperacion.text = resuString
}
```

```
    @IBAction func btnMulti(_ sender: UIButton) {
        if num == 0 {
            num = Int(String(labelNumero.text!))!
            resuString += String(num)
        }
        else{
            num2 = Int(String(labelNumero.text!))!
            resultado += CMatematicas.Multi(n1: num, n2: num2)

            resuString += " * " + String(num2)
        }
        txtOperacion.text = resuString
    }


    @IBAction func btnModulo(_ sender: UIButton) {
        if num == 0 {
            num = Int(String(labelNumero.text!))!

            resuString += String(num)

        }
        else{
            num2 = Int(String(labelNumero.text!))!
            resultado += CMatematicas.Mod(n1: num, n2: num2)

            resuString += " % " + String(num2)

        }
        txtOperacion.text = resuString
    }

    @IBAction func btnIgual(_ sender: UIButton) {
        resuString += " = " + String(resultado)
        labelNumero.text = String(resultado)
        txtOperacion.text = resuString
        resuString = ""
    }
}
```

Vista2VC.swift

```
import UIKit

class Vista2VC: UIViewController {
```

```
@IBOutlet weak var txtConjuntoA: UITextField!
@IBOutlet weak var txtConjuntoB: UITextField!
@IBOutlet weak var segmentoConjunto: UISegmentedControl!
@IBOutlet weak var lblResultado: UILabel!

@IBOutlet weak var btnCalcularOutlet: UIButton!
var conjuntoA:Set<String> = []
var conjuntoB:Set<String> = []
var opc: Int = 0

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
}

@IBAction func segmentedAction(_ sender: UISegmentedControl) {
    let indice : Int = sender.selectedSegmentIndex

    opc = indice
}


@IBAction func txtConjuntoAAction(_ sender: UITextField) {
    txtConjuntoA.becomeFirstResponder()

    if !txtConjuntoA.text!.isEmpty && !txtConjuntoB.text!.isEmpty   {
        btnCalcularOutlet.isEnabled = true
    }
    else {
        btnCalcularOutlet.isEnabled = false
    }

 }

@IBAction func txtConjuntoBAction(_ sender: UITextField) {
    txtConjuntoB.becomeFirstResponder()

    if !txtConjuntoA.text!.isEmpty && !txtConjuntoB.text!.isEmpty  {
        btnCalcularOutlet.isEnabled = true
    }
    else {
        btnCalcularOutlet.isEnabled = false
    }
}

@IBAction func btnCalcular(_ sender: UIButton) {
```

```
        let strA = txtConjuntoA.text
        let strB = txtConjuntoB.text
        // numeros enteros INT
       // conjuntoA = deStringAConjunto(conjunto: conjuntoA, cadena: strA!)
        //conjuntoB = deStringAConjunto(conjunto: conjuntoB, cadena: strB!)
        // cadenas de texto STRING
        conjuntoA = deStringAConjuntoStr(conjunto: conjuntoA, cadena: strA!)
         conjuntoB = deStringAConjuntoStr(conjunto: conjuntoB, cadena: strB!)

        switch(opc){
        case 0: // Union
            let unionAB = conjuntoA.union(conjuntoB).sorted()
            let resultado = unionAB
            lblResultado.text =  "A:\(conjuntoA) U B:\(conjuntoB) =
\(resultado.joined(separator: ","))"
            break;
        case 1:
            let interseccionAB = conjuntoA.intersection(conjuntoB).sorted()
            let resultado = interseccionAB
            lblResultado.text = "A:\(conjuntoA) ∩ B:\(conjuntoB) = \(resultado.joined(separator:
","))"
            break;
        case 2:
            let diferenciaAB = conjuntoA.subtracting(conjuntoB).sorted()
            let resultado = diferenciaAB
            lblResultado.text = "A:\(conjuntoA) - B:\(conjuntoB) = \(resultado.joined(separator:
","))"
            break;
        case 3:
            let diferenciaSimetricaAB = conjuntoA.symmetricDifference(conjuntoB).sorted()
            let resultado = diferenciaSimetricaAB
            lblResultado.text = "A: \(conjuntoA.sorted()) ∆ B:\(conjuntoB) =
\(resultado.joined(separator: ","))"
            break;
        default:
            lblResultado.text = "A: \(conjuntoA.sorted()), B: \(conjuntoB.sorted())"
        }


    }

    func deStringAConjunto(conjunto:Set<Int>, cadena: String) -> Set<Int>{
        var conjuntoAux:Set<Int> = []
        var indice = 0
        var num: String = ""
        for caracter in cadena {
            indice += 1
```

```
        if caracter >= "0" && caracter <= "9" {
            num += String(caracter)

            if indice == (cadena.count){
                conjuntoAux.insert(Int(num)!)
                num = ""
            }

        }else if caracter == "," {
            conjuntoAux.insert(Int(num)!)
            num = ""

        }else if indice == (cadena.count) - 1 {
            conjuntoAux.insert(Int(num)!)
            num = ""
        }
    }
    return conjuntoAux
}
func deStringAConjuntoStr(conjunto:Set<String>, cadena: String) -> Set<String>{
    var conjuntoAux:Set<String> = []
    var indice = 0
    var num: String = ""
    for caracter in cadena {
        indice += 1
        if caracter != "," {
        num += String(caracter)

        if indice == (cadena.count){
            conjuntoAux.insert(num)
            num = ""
        }

        }else if caracter == "," {
            conjuntoAux.insert(num)
            num = ""

        }
    }
    conjuntoAux.remove(" ")
    return conjuntoAux
}


/*
// MARK: - Navigation
```

```
    // In a storyboard-based application, you will often want to do a little preparation before
navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        // Get the new view controller using segue.destination.
        // Pass the selected object to the new view controller.
    }
    */

}
```

Vista4VC.swift

```
import UIKit

class Vista4VC: UIViewController, UIImagePickerControllerDelegate,
UINavigationControllerDelegate {

    @IBOutlet weak var imgViewFotos: UIImageView!

    let imgPicker = UIImagePickerController()

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
        imgPicker.delegate = self

        // Agregar una animacion
        self.imgViewFotos.alpha = 0.0

        UIImageView.animate(withDuration: 2){
            self.imgViewFotos.frame = CGRect(x: 0, y: 0, width: 370, height: 460)
            self.imgViewFotos.center = self.view.center
            self.imgViewFotos.alpha = 1.0
            self.imgViewFotos.layer.cornerRadius = 50
        }
    }

    @IBAction func btnSeleccionarFotos(_ sender: UIButton) {
        imgPicker.allowsEditing = false
        imgPicker.sourceType = .photoLibrary //.camera

        present(imgPicker, animated: true,completion: nil)
    }

    func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
```

```
        if let pickerImage = info[UIImagePickerController.InfoKey.originalImage] as? UIImage
{
            imgViewFotos.contentMode = .scaleAspectFit
            imgViewFotos.image = pickerImage
        }
        dismiss(animated: true, completion: nil)
    }


}
```

Vista5VC.swift

```
import UIKit
import SceneKit

class Vista5VC: UIViewController {

    @IBOutlet weak var imageView: UIImageView!
    @IBOutlet weak var segmentoLuces: UISegmentedControl!
    @IBOutlet weak var segmentoFiguras: UISegmentedControl!

    @IBOutlet weak var sliderLuzX: UISlider!
    @IBOutlet weak var sliderLuzY: UISlider!
    @IBOutlet weak var sliderLuzZ: UISlider!

    @IBOutlet weak var sliderCamaraX: UISlider!
    @IBOutlet weak var sliderCamaraY: UISlider!
    @IBOutlet weak var sliderCamaraZ: UISlider!

    var tipoLuz : String? = "ambient"
    var tipoFigura : Int = 0

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }
    // LUZ
    @IBAction func sliderLuzX(_ sender: UISlider) {
        sliderLuzX.value = sender.value
        seleccFigura()

    }
    @IBAction func sliderLuzY(_ sender: UISlider) {
        sliderLuzY.value = sender.value
        seleccFigura()
```

```
}
@IBAction func sliderLuzZ(_ sender: UISlider) {
    sliderLuzZ.value = sender.value
    seleccFigura()
}

// CAMARA
@IBAction func sliderCamaraX(_ sender: UISlider) {
    sliderCamaraX.value = sender.value
    seleccFigura()
}
@IBAction func sliderCamaraY(_ sender: UISlider) {
    sliderCamaraY.value = sender.value
    seleccFigura()
}
@IBAction func sliderCamaraZ(_ sender: UISlider) {
    sliderCamaraZ.value = sender.value
    seleccFigura()
}

// SEGMENTED
@IBAction func segmentoLuces(_ sender: UISegmentedControl) {
    let indice : Int = segmentoLuces.selectedSegmentIndex

    switch indice {
    case 0: tipoLuz = "ambient"
    case 1: tipoLuz = "directional"
    case 2: tipoLuz = "omni"
    case 3: tipoLuz = "spot"
    case 4: tipoLuz = "IES"
    case 5: tipoLuz = "probe"
    case 6: tipoLuz = "area"
    default:tipoLuz = "ambient"
    }

    seleccFigura()
}


@IBAction func segmentoFiguras(_ sender: UISegmentedControl) {
    let indice : Int = segmentoFiguras.selectedSegmentIndex

    switch indice {
    case 0: tipoFigura = 0
    case 1: tipoFigura = 1
    case 2: tipoFigura = 2
    case 3: tipoFigura = 3
```

```
        case 4: tipoFigura = 4
        case 5: tipoFigura = 5
        case 6: tipoFigura = 6
        default:tipoFigura = 0
        }

        seleccFigura()
    }

    func seleccFigura(){
        switch tipoFigura {
        case 0: dibujaCubo()
        case 1: dibujaPyramid()
        case 2: dibujaCylinder()
        case 3: dibujaTube()
        case 4: dibujaTorus()
        case 5: dibujaCone()
        case 6: dibujaSphere()
        default: dibujaCubo()
        }
    }

    func dibujaCubo(){
        let sceneView = SCNView(frame: self.imageView.frame)
        self.imageView.addSubview(sceneView)
        let scene = SCNScene()
        sceneView.scene = scene

        let camara = SCNCamera()
        let camaraNodo = SCNNode()
        camaraNodo.camera = camara

        // Vista de frente
        //camaraNodo.position = SCNVector3(x: 0.0, y: 0, z: 3.0)
        camaraNodo.position = SCNVector3(x: Float(sliderCamaraX.value),
y:Float(sliderCamaraY.value), z:Float(sliderCamaraZ.value))

        let luz = SCNLight()
        //luz.type = SCNLight.LightType.spot
        luz.type = SCNLight.LightType(rawValue: tipoLuz!)

        luz.spotInnerAngle = 30.0 // DEBE SER MODIFICABLE
        luz.spotOuterAngle = 70.0 // DEBE SER MODIFICABLE
        luz.castsShadow = true

        let luzNodo = SCNNode()
        luzNodo.light = luz
```

```
        // Establecer los max y mins de los sliders
        luzNodo.position = SCNVector3(x: sliderLuzX.value, y:sliderLuzY.value,
z:sliderLuzZ.value)

        //Cubo
        let geometriaCubo = SCNBox(width: 1.0, height: 1.0, length: 1.0, chamferRadius: 0.2)

        let cuboNodo = SCNNode(geometry: geometriaCubo)

        let constraint = SCNLookAtConstraint(target: cuboNodo)

        constraint.isGimbalLockEnabled = true
        camaraNodo.constraints = [constraint]
        luzNodo.constraints = [constraint]

        // PARA LAS SOMBRAS Y EL MATERIAL
        // HACER MODIFICACIONES
        let planoGeometria = SCNPlane(width: 50.0, height: 50.0)
        let planoNodo = SCNNode(geometry: planoGeometria)

        // Modifiar el valor angular de los 3 ejes
        planoNodo.eulerAngles = SCNVector3(x:GLKMathDegreesToRadians(-90), y: 0, z: 0)

        planoNodo.position = SCNVector3(x:0.0, y:-0.5, z:0.0)

        // Color cubo
        let materialCubo1 = SCNMaterial()
        materialCubo1.diffuse.contents = UIColor.systemGreen
        geometriaCubo.materials = [materialCubo1]

        // Color plano cubo
        let materialCubo2 = SCNMaterial()
        materialCubo2.diffuse.contents = UIColor.systemYellow
        planoGeometria.materials = [materialCubo2]

        // Utilizamos los nodos
        scene.rootNode.addChildNode(luzNodo)
        scene.rootNode.addChildNode(camaraNodo)
        scene.rootNode.addChildNode(cuboNodo)
        scene.rootNode.addChildNode(planoNodo)

    }

    func dibujaPyramid(){
        let sceneView = SCNView(frame: self.imageView.frame)
        self.imageView.addSubview(sceneView)
        let scene = SCNScene()
```

```
        sceneView.scene = scene

        let camara = SCNCamera()
        let camaraNodo = SCNNode()
        camaraNodo.camera = camara

        // Vista de frente
        //camaraNodo.position = SCNVector3(x: 0.0, y: 0, z: 3.0)
        camaraNodo.position = SCNVector3(x: Float(sliderCamaraX.value),
y:Float(sliderCamaraY.value), z:Float(sliderCamaraZ.value))

        let luz = SCNLight()
        //luz.type = SCNLight.LightType.spot
        luz.type = SCNLight.LightType(rawValue: tipoLuz!)

        luz.spotInnerAngle = 30.0 // DEBE SER MODIFICABLE
        luz.spotOuterAngle = 70.0 // DEBE SER MODIFICABLE
        luz.castsShadow = true

        let luzNodo = SCNNode()
        luzNodo.light = luz
        // Establecer los max y mins de los sliders
        luzNodo.position = SCNVector3(x: sliderLuzX.value, y:sliderLuzY.value,
z:sliderLuzZ.value)

        // Piramide
        let geometriaCubo = SCNPyramid(width: 1.0, height: 1.0, length: 1.0)
        // Cilindro
        //let geometriaCubo = SCNCylinder(radius: 0.5, height: 1.0)
        // Tubo
        //let geometriaCubo = SCNTube(innerRadius: 0.0, outerRadius: 1.0, height: 1.0)
        // Dona
        //let geometriaCubo = SCNTorus(ringRadius: 1.0, pipeRadius: 0.5)

        //let geometriaCubo = SCNCone(topRadius: 0.0, bottomRadius: 0.7, height: 1.0)

        //let geometriaCubo = SCNSphere(radius: 0.7)

        let cuboNodo = SCNNode(geometry: geometriaCubo)

        let constraint = SCNLookAtConstraint(target: cuboNodo)

        constraint.isGimbalLockEnabled = true
        camaraNodo.constraints = [constraint]
        luzNodo.constraints = [constraint]

        // PARA LAS SOMBRAS Y EL MATERIAL
```

```
        // HACER MODIFICACIONES
        let planoGeometria = SCNPlane(width: 50.0, height: 50.0)
        let planoNodo = SCNNode(geometry: planoGeometria)

        // Modifiar el valor angular de los 3 ejes
        planoNodo.eulerAngles = SCNVector3(x:GLKMathDegreesToRadians(-90), y: 0, z: 0)

        planoNodo.position = SCNVector3(x:0.0, y:-0.5, z:0.0)

        // Color cubo
        let materialCubo1 = SCNMaterial()
        materialCubo1.diffuse.contents = UIColor.systemGreen
        geometriaCubo.materials = [materialCubo1]

        // Color plano cubo
        let materialCubo2 = SCNMaterial()
        materialCubo2.diffuse.contents = UIColor.systemYellow
        planoGeometria.materials = [materialCubo2]

        // Utilizamos los nodos
        scene.rootNode.addChildNode(luzNodo)
        scene.rootNode.addChildNode(camaraNodo)
        scene.rootNode.addChildNode(cuboNodo)
        scene.rootNode.addChildNode(planoNodo)

    }

    func dibujaCylinder(){
        let sceneView = SCNView(frame: self.imageView.frame)
        self.imageView.addSubview(sceneView)
        let scene = SCNScene()
        sceneView.scene = scene

        let camara = SCNCamera()
        let camaraNodo = SCNNode()
        camaraNodo.camera = camara

        // Vista de frente
        //camaraNodo.position = SCNVector3(x: 0.0, y: 0, z: 3.0)
        camaraNodo.position = SCNVector3(x: Float(sliderCamaraX.value),
y:Float(sliderCamaraY.value), z:Float(sliderCamaraZ.value))

        let luz = SCNLight()
        //luz.type = SCNLight.LightType.spot
        luz.type = SCNLight.LightType(rawValue: tipoLuz!)

        luz.spotInnerAngle = 30.0 // DEBE SER MODIFICABLE
```

```
        luz.spotOuterAngle = 70.0 // DEBE SER MODIFICABLE
        luz.castsShadow = true

        let luzNodo = SCNNode()
        luzNodo.light = luz
        // Establecer los max y mins de los sliders
        luzNodo.position = SCNVector3(x: sliderLuzX.value, y:sliderLuzY.value,
z:sliderLuzZ.value)

        // Cilindro
        let geometriaCubo = SCNCylinder(radius: 0.5, height: 1.0)
        // Tubo
        //let geometriaCubo = SCNTube(innerRadius: 0.0, outerRadius: 1.0, height: 1.0)
        // Dona
        //let geometriaCubo = SCNTorus(ringRadius: 1.0, pipeRadius: 0.5)

        //let geometriaCubo = SCNCone(topRadius: 0.0, bottomRadius: 0.7, height: 1.0)

        //let geometriaCubo = SCNSphere(radius: 0.7)

        let cuboNodo = SCNNode(geometry: geometriaCubo)

        let constraint = SCNLookAtConstraint(target: cuboNodo)

        constraint.isGimbalLockEnabled = true
        camaraNodo.constraints = [constraint]
        luzNodo.constraints = [constraint]

        // PARA LAS SOMBRAS Y EL MATERIAL
        // HACER MODIFICACIONES
        let planoGeometria = SCNPlane(width: 50.0, height: 50.0)
        let planoNodo = SCNNode(geometry: planoGeometria)

        // Modifiar el valor angular de los 3 ejes
        planoNodo.eulerAngles = SCNVector3(x:GLKMathDegreesToRadians(-90), y: 0, z: 0)

        planoNodo.position = SCNVector3(x:0.0, y:-0.5, z:0.0)

        // Color cubo
        let materialCubo1 = SCNMaterial()
        materialCubo1.diffuse.contents = UIColor.systemGreen
        geometriaCubo.materials = [materialCubo1]

        // Color plano cubo
        let materialCubo2 = SCNMaterial()
        materialCubo2.diffuse.contents = UIColor.systemYellow
        planoGeometria.materials = [materialCubo2]
```

```
        // Utilizamos los nodos
        scene.rootNode.addChildNode(luzNodo)
        scene.rootNode.addChildNode(camaraNodo)
        scene.rootNode.addChildNode(cuboNodo)
        scene.rootNode.addChildNode(planoNodo)

    }

    func dibujaTube(){
        let sceneView = SCNView(frame: self.imageView.frame)
        self.imageView.addSubview(sceneView)
        let scene = SCNScene()
        sceneView.scene = scene

        let camara = SCNCamera()
        let camaraNodo = SCNNode()
        camaraNodo.camera = camara

        // Vista de frente
        //camaraNodo.position = SCNVector3(x: 0.0, y: 0, z: 3.0)
        camaraNodo.position = SCNVector3(x: Float(sliderCamaraX.value),
y:Float(sliderCamaraY.value), z:Float(sliderCamaraZ.value))

        let luz = SCNLight()
        //luz.type = SCNLight.LightType.spot
        luz.type = SCNLight.LightType(rawValue: tipoLuz!)

        luz.spotInnerAngle = 30.0 // DEBE SER MODIFICABLE
        luz.spotOuterAngle = 70.0 // DEBE SER MODIFICABLE
        luz.castsShadow = true

        let luzNodo = SCNNode()
        luzNodo.light = luz
        // Establecer los max y mins de los sliders
        luzNodo.position = SCNVector3(x: sliderLuzX.value, y:sliderLuzY.value,
z:sliderLuzZ.value)

        // Tubo
        let geometriaCubo = SCNTube(innerRadius: 0.0, outerRadius: 1.0, height: 1.0)
        // Dona
        //let geometriaCubo = SCNTorus(ringRadius: 1.0, pipeRadius: 0.5)

        //let geometriaCubo = SCNCone(topRadius: 0.0, bottomRadius: 0.7, height: 1.0)

        //let geometriaCubo = SCNSphere(radius: 0.7)
```

```
    let cuboNodo = SCNNode(geometry: geometriaCubo)

    let constraint = SCNLookAtConstraint(target: cuboNodo)

    constraint.isGimbalLockEnabled = true
    camaraNodo.constraints = [constraint]
    luzNodo.constraints = [constraint]

    // PARA LAS SOMBRAS Y EL MATERIAL
    // HACER MODIFICACIONES
    let planoGeometria = SCNPlane(width: 50.0, height: 50.0)
    let planoNodo = SCNNode(geometry: planoGeometria)

    // Modifiar el valor angular de los 3 ejes
    planoNodo.eulerAngles = SCNVector3(x:GLKMathDegreesToRadians(-90), y: 0, z: 0)

    planoNodo.position = SCNVector3(x:0.0, y:-0.5, z:0.0)

    // Color cubo
    let materialCubo1 = SCNMaterial()
    materialCubo1.diffuse.contents = UIColor.systemGreen
    geometriaCubo.materials = [materialCubo1]

    // Color plano cubo
    let materialCubo2 = SCNMaterial()
    materialCubo2.diffuse.contents = UIColor.systemYellow
    planoGeometria.materials = [materialCubo2]

    // Utilizamos los nodos
    scene.rootNode.addChildNode(luzNodo)
    scene.rootNode.addChildNode(camaraNodo)
    scene.rootNode.addChildNode(cuboNodo)
    scene.rootNode.addChildNode(planoNodo)

}

func dibujaTorus(){
    let sceneView = SCNView(frame: self.imageView.frame)
    self.imageView.addSubview(sceneView)
    let scene = SCNScene()
    sceneView.scene = scene

    let camara = SCNCamera()
    let camaraNodo = SCNNode()
    camaraNodo.camera = camara

    // Vista de frente
```

```
        //camaraNodo.position = SCNVector3(x: 0.0, y: 0, z: 3.0)
        camaraNodo.position = SCNVector3(x: Float(sliderCamaraX.value),
y:Float(sliderCamaraY.value), z:Float(sliderCamaraZ.value))

        let luz = SCNLight()
        //luz.type = SCNLight.LightType.spot
        luz.type = SCNLight.LightType(rawValue: tipoLuz!)

        luz.spotInnerAngle = 30.0 // DEBE SER MODIFICABLE
        luz.spotOuterAngle = 70.0 // DEBE SER MODIFICABLE
        luz.castsShadow = true

        let luzNodo = SCNNode()
        luzNodo.light = luz
        // Establecer los max y mins de los sliders
        luzNodo.position = SCNVector3(x: sliderLuzX.value, y:sliderLuzY.value,
z:sliderLuzZ.value)

        // Dona
        let geometriaCubo = SCNTorus(ringRadius: 1.0, pipeRadius: 0.5)

        //let geometriaCubo = SCNCone(topRadius: 0.0, bottomRadius: 0.7, height: 1.0)

        //let geometriaCubo = SCNSphere(radius: 0.7)

        let cuboNodo = SCNNode(geometry: geometriaCubo)

        let constraint = SCNLookAtConstraint(target: cuboNodo)

        constraint.isGimbalLockEnabled = true
        camaraNodo.constraints = [constraint]
        luzNodo.constraints = [constraint]

        // PARA LAS SOMBRAS Y EL MATERIAL
        // HACER MODIFICACIONES
        let planoGeometria = SCNPlane(width: 50.0, height: 50.0)
        let planoNodo = SCNNode(geometry: planoGeometria)

        // Modifiar el valor angular de los 3 ejes
        planoNodo.eulerAngles = SCNVector3(x:GLKMathDegreesToRadians(-90), y: 0, z: 0)

        planoNodo.position = SCNVector3(x:0.0, y:-0.5, z:0.0)

        // Color cubo
        let materialCubo1 = SCNMaterial()
        materialCubo1.diffuse.contents = UIColor.systemGreen
        geometriaCubo.materials = [materialCubo1]
```

```
        // Color plano cubo
        let materialCubo2 = SCNMaterial()
        materialCubo2.diffuse.contents = UIColor.systemYellow
        planoGeometria.materials = [materialCubo2]

        // Utilizamos los nodos
        scene.rootNode.addChildNode(luzNodo)
        scene.rootNode.addChildNode(camaraNodo)
        scene.rootNode.addChildNode(cuboNodo)
        scene.rootNode.addChildNode(planoNodo)

    }

    func dibujaCone(){
        let sceneView = SCNView(frame: self.imageView.frame)
        self.imageView.addSubview(sceneView)
        let scene = SCNScene()
        sceneView.scene = scene

        let camara = SCNCamera()
        let camaraNodo = SCNNode()
        camaraNodo.camera = camara

        // Vista de frente
        //camaraNodo.position = SCNVector3(x: 0.0, y: 0, z: 3.0)
        camaraNodo.position = SCNVector3(x: Float(sliderCamaraX.value),
y:Float(sliderCamaraY.value), z:Float(sliderCamaraZ.value))

        let luz = SCNLight()
        //luz.type = SCNLight.LightType.spot
        luz.type = SCNLight.LightType(rawValue: tipoLuz!)

        luz.spotInnerAngle = 30.0 // DEBE SER MODIFICABLE
        luz.spotOuterAngle = 70.0 // DEBE SER MODIFICABLE
        luz.castsShadow = true

        let luzNodo = SCNNode()
        luzNodo.light = luz
        // Establecer los max y mins de los sliders
        luzNodo.position = SCNVector3(x: sliderLuzX.value, y:sliderLuzY.value,
z:sliderLuzZ.value)


        // Cono
        let geometriaCubo = SCNCone(topRadius: 0.0, bottomRadius: 0.7, height: 1.0)
```

```
    //let geometriaCubo = SCNSphere(radius: 0.7)

    let cuboNodo = SCNNode(geometry: geometriaCubo)

    let constraint = SCNLookAtConstraint(target: cuboNodo)

    constraint.isGimbalLockEnabled = true
    camaraNodo.constraints = [constraint]
    luzNodo.constraints = [constraint]

    // PARA LAS SOMBRAS Y EL MATERIAL
    // HACER MODIFICACIONES
    let planoGeometria = SCNPlane(width: 50.0, height: 50.0)
    let planoNodo = SCNNode(geometry: planoGeometria)

    // Modifiar el valor angular de los 3 ejes
    planoNodo.eulerAngles = SCNVector3(x:GLKMathDegreesToRadians(-90), y: 0, z: 0)

    planoNodo.position = SCNVector3(x:0.0, y:-0.5, z:0.0)

    // Color cubo
    let materialCubo1 = SCNMaterial()
    materialCubo1.diffuse.contents = UIColor.systemGreen
    geometriaCubo.materials = [materialCubo1]

    // Color plano cubo
    let materialCubo2 = SCNMaterial()
    materialCubo2.diffuse.contents = UIColor.systemYellow
    planoGeometria.materials = [materialCubo2]

    // Utilizamos los nodos
    scene.rootNode.addChildNode(luzNodo)
    scene.rootNode.addChildNode(camaraNodo)
    scene.rootNode.addChildNode(cuboNodo)
    scene.rootNode.addChildNode(planoNodo)

}

func dibujaSphere(){
    let sceneView = SCNView(frame: self.imageView.frame)
    self.imageView.addSubview(sceneView)
    let scene = SCNScene()
    sceneView.scene = scene

    let camara = SCNCamera()
    let camaraNodo = SCNNode()
    camaraNodo.camera = camara
```

```
        // Vista de frente
        //camaraNodo.position = SCNVector3(x: 0.0, y: 0, z: 3.0)
        camaraNodo.position = SCNVector3(x: Float(sliderCamaraX.value),
y:Float(sliderCamaraY.value), z:Float(sliderCamaraZ.value))

        let luz = SCNLight()
        //luz.type = SCNLight.LightType.spot
        luz.type = SCNLight.LightType(rawValue: tipoLuz!)

        luz.spotInnerAngle = 30.0 // DEBE SER MODIFICABLE
        luz.spotOuterAngle = 70.0 // DEBE SER MODIFICABLE
        luz.castsShadow = true

        let luzNodo = SCNNode()
        luzNodo.light = luz
        // Establecer los max y mins de los sliders
        luzNodo.position = SCNVector3(x: sliderLuzX.value, y:sliderLuzY.value,
z:sliderLuzZ.value)


        // Esfera
        let geometriaCubo = SCNSphere(radius: 0.7)

        let cuboNodo = SCNNode(geometry: geometriaCubo)

        let constraint = SCNLookAtConstraint(target: cuboNodo)

        constraint.isGimbalLockEnabled = true
        camaraNodo.constraints = [constraint]
        luzNodo.constraints = [constraint]

        // PARA LAS SOMBRAS Y EL MATERIAL
        // HACER MODIFICACIONES
        let planoGeometria = SCNPlane(width: 50.0, height: 50.0)
        let planoNodo = SCNNode(geometry: planoGeometria)

        // Modifiar el valor angular de los 3 ejes
        planoNodo.eulerAngles = SCNVector3(x:GLKMathDegreesToRadians(-90), y: 0, z: 0)

        planoNodo.position = SCNVector3(x:0.0, y:-0.5, z:0.0)

        // Color cubo
        let materialCubo1 = SCNMaterial()
        materialCubo1.diffuse.contents = UIColor.systemGreen
        geometriaCubo.materials = [materialCubo1]
```

```
        // Color plano cubo
        let materialCubo2 = SCNMaterial()
        materialCubo2.diffuse.contents = UIColor.systemYellow
        planoGeometria.materials = [materialCubo2]

        // Utilizamos los nodos
        scene.rootNode.addChildNode(luzNodo)
        scene.rootNode.addChildNode(camaraNodo)
        scene.rootNode.addChildNode(cuboNodo)
        scene.rootNode.addChildNode(planoNodo)

    }

    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a little preparation before
navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        // Get the new view controller using segue.destination.
        // Pass the selected object to the new view controller.
    }
    */

}
```

Vista6VC.swift

```
import UIKit
import SceneKit

class Vista6VC: UIViewController {

    @IBOutlet weak var imageView: UIImageView!

    @IBOutlet weak var Iniciar: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.

    }
    @IBAction func Iniciar(_ sender: UIButton) {
        Malla()
    }
```

```
func sinFunction(x: Float,z: Float) -> Float {
  return 0.2 * sin(x * 5 + z * 3) + 0.1 * cos(x * 5 + z * 10 + 0.6) + 0.05 * cos(x * x * z)
}

func squareFunction(x: Float,z: Float) -> Float {
  return x * x + z * z
}

func Malla(){
   let sceneView = SCNView(frame: self.imageView.frame)

   self.imageView.addSubview(sceneView)
   let scene = SCNScene()
   sceneView.scene = scene

   let camara = SCNCamera()
   let camaraNodo = SCNNode()
   camaraNodo.camera = camara

   let gridSize = 40

   let capsuleRadius:CGFloat = 1.0 / CGFloat(gridSize - 1)
   let capsuleHeight:CGFloat = capsuleRadius * 6.0

   var z:Float = Float(-gridSize + 1) * Float(capsuleRadius)

   for _ in 0..<gridSize {
     var x:Float = Float(-gridSize + 1) * Float(capsuleRadius)
       for _ in 0..<gridSize {
         let capsule = SCNCapsule(capRadius: capsuleRadius, height: capsuleHeight)

         let hue = CGFloat(abs(x * z))
         let color = UIColor(hue: hue, saturation: 1.0, brightness: 1.0, alpha: 1.0)

         capsule.firstMaterial?.diffuse.contents = color

         let capsuleNode = SCNNode(geometry: capsule)

         scene.rootNode.addChildNode(capsuleNode)


         capsuleNode.position = SCNVector3Make(x, 2.0, z)

         let y = CGFloat(squareFunction(x: x, z: z))
       // let y = CGFloat(sinFunction(x: x, z: z))
```

```
        let moveUp = SCNAction.moveBy(x: 0, y: y, z: 0, duration: 2.0)
        let moveDown = SCNAction.moveBy(x: 0, y: -y, z: 0, duration: 2.0)

        let sequence = SCNAction.sequence([moveUp,moveDown])

        let repeatedSequence = SCNAction.repeatForever(sequence)

        capsuleNode.runAction(repeatedSequence)

        x += 2.0 * Float(capsuleRadius)
      }

      z += 2.0 * Float(capsuleRadius)

      for _ in 0..<gridSize {
        let capsule = SCNCapsule(capRadius: capsuleRadius, height: capsuleHeight)

        let hue = CGFloat(abs(x * z))
        let color = UIColor(hue: hue, saturation: 1.0, brightness: 1.0, alpha: 1.0)

        capsule.firstMaterial?.diffuse.contents = color

        let capsuleNode = SCNNode(geometry: capsule)

        scene.rootNode.addChildNode(capsuleNode)


        capsuleNode.position = SCNVector3Make(x, 2.0, z)

       // let y = CGFloat(squareFunction(x: x, z: z))
       let y = CGFloat(sinFunction(x: x, z: z))

        let moveUp = SCNAction.moveBy(x: 0, y: y, z: 0, duration: 2.0)
        let moveDown = SCNAction.moveBy(x: 0, y: -y, z: 0, duration: 2.0)

        let sequence = SCNAction.sequence([moveUp,moveDown])

        let repeatedSequence = SCNAction.repeatForever(sequence)

        capsuleNode.runAction(repeatedSequence)

        x += 2.0 * Float(capsuleRadius)

      }

      z += 2.0 * Float(capsuleRadius)
    }
```

```
    }
}
```

appDireccion

ContentView.swift

```
import MapKit
import SwiftUI
import UIKit

struct ContentView: View {

  @State private var directions: [String] = []
  @State private var showDirections = false

  var body: some View {
    VStack {
      MapView(directions: $directions)

      Button(action: {
        self.showDirections.toggle()
      }, label: {
        Text("Mostra Indicaciones")
      })
      .disabled(directions.isEmpty)
      .padding()
    }.sheet(isPresented: $showDirections, content: {
      VStack(spacing: 0) {
        Text("Indicaciones para llegar al tec")
          .font(.largeTitle)
          .bold()
          .padding()

        Divider().background(Color(UIColor.systemBlue))

        List(0..<self.directions.count, id: \.self) { i in
          Text(self.directions[i]).padding()
        }
      }
    })
  }
}

struct MapView: UIViewRepresentable {
```

```
  typealias UIViewType = MKMapView

  @Binding var directions: [String]

  func makeCoordinator() -> MapViewCoordinator {
    return MapViewCoordinator()
  }

  func makeUIView(context: Context) -> MKMapView {
    let mapView = MKMapView()
    mapView.delegate = context.coordinator

    let region = MKCoordinateRegion(
      center: CLLocationCoordinate2D(latitude: 25.560943344280517, longitude: -
103.39759393033292),
      span: MKCoordinateSpan(latitudeDelta: 0.03, longitudeDelta: 0.03))
    mapView.setRegion(region, animated: true)

    // llegar de mi casa al tec
    // Cine Tec
      let p1 = MKPlacemark(coordinate: CLLocationCoordinate2D(latitude:
25.560943344280517, longitude: -103.39759393033292))

    // Yo
    let p2 = MKPlacemark(coordinate: CLLocationCoordinate2D(latitude:
25.533260932165902, longitude: -103.43562389710982))


    let request = MKDirections.Request()
    request.source = MKMapItem(placemark: p1)
    request.destination = MKMapItem(placemark: p2)
    request.transportType = .automobile

    let directions = MKDirections(request: request)
    directions.calculate { response, error in
      guard let route = response?.routes.first else { return }
      //mapView.addAnnotations([p1, p2])
      mapView.addOverlay(route.polyline)
      mapView.setVisibleMapRect(
        route.polyline.boundingMapRect,
        edgePadding: UIEdgeInsets(top: 10, left: 10, bottom: 10, right: 10),
        animated: true)
      self.directions = route.steps.map { $0.instructions }.filter { !$0.isEmpty }
    }
    return mapView
  }
```

```
  func updateUIView(_ uiView: MKMapView, context: Context) {
  }

  class MapViewCoordinator: NSObject, MKMapViewDelegate {
    func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) ->
MKOverlayRenderer {
      let renderer = MKPolylineRenderer(overlay: overlay)
      renderer.strokeColor = .systemBlue
      renderer.lineWidth = 5
      return renderer
    }
  }
}

struct ContentView_Previews: PreviewProvider {
  static var previews: some View {
    ContentView()
  }
}
```

DirectionsApp.swift

```
import SwiftUI

@main
struct DirectionsApp: App {
  var body: some Scene {
    WindowGroup {
      ContentView()
    }
  }
}
```

Referencias:

Cairocoders. (2021, 4 junio). SwiftUI Firestore CRUD Create, Read, Update and Delete. YouTube. https://www.youtube.com/watch?v=w7zwyOW83HU

Ale Patron. (2020, 18 septiembre). SwiftUI Tutorial - MapKit, Route, and Directions. YouTube. https://www.youtube.com/watch?v=H6pmm62axCg