

II Tinder

Add a file + Add a View (xib) \Rightarrow size: free form class: CardView

II CardView.swift

```
import UIKit
```

```
class CardView: UIView {
```

```
    override func didMoveToSuperview() {
        super.didMoveToSuperview()
        layer.shadowColor = UIColor.black.cgColor
        layer.shadowOpacity = 0.4
        layer.shadowOffset = CGSize.zero
        layer.shadowRadius = CGFloat(12)
    }
```

```
    override func layoutSubviews() {
        super.layoutSubviews()
        layer.cornerRadius = CGFloat(6)
    }
}
```

II ViewController

```
import UIKit
```

```
class ViewController: UIViewController {
```

```
    var cardView: CardView!
```

```
    enum SwipeDirection: String {
```

```
        case Left, Right
    }
```

```
    lazy var panRecognizer: UIPanGestureRecognizer = {
        let recognizer = UIPanGestureRecognizer()
        recognizer.addTarget(self, action: #selector(cardViewPanned(recognizer:)))
        return recognizer
    }()

```

```
    var animator: UIViewPropertyAnimator = UIViewPropertyAnimator()
```

```
    var animateProgress: CGFloat = 0
```

```
    var currentSwipeDirection: SwipeDirection = .Right
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

```
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        createNewCard()
    }
}
```

```

objc func cardViewPaned(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
        case .began:
            let translation = recognizer.translation(in: cardView)
            if translation.x > 0 {
                animateSwipe(direction: .right)
            } else if translation.x < 0 {
                animateSwipe(direction: .left)
            }
        }
        animator.pauseAnimation()
        animateProgress = animator.fractionComplete
        case .changed:
            let translation = recognizer.translation(in: cardView)
            var fraction = translation.x / (view.frame.width)
            if currentSwipeDirection == .left {
                fraction *= -1
            }
        }
        animator.fractionComplete = fraction + animateProgress
        if animator.fractionComplete == CGFloat(1.0) {
            if currentSwipeDirection == .left && translation.x > 0 {
                refreshAnimator(direction: .right)
            } else if currentSwipeDirection == .right && translation.x < 0 {
                refreshAnimator(direction: .left)
            }
        }
    }
    case .ended:
        let velocity = recognizer.velocity(in: cardView)
        if velocity.x > 100 || animator.fractionComplete > 0.6 || velocity.x < -100 {
            animator.addCompletion { [position] in
                self.cardView.removeFromSuperview()
                self.createNewCard()
            }
            animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
            break
        }
        animator.isReversed = true
        animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
    }
    default:
        break
    }
}

```

```
func animateSwipe (direction : SwipeDirection) {
    if animator.isRunning { return }
    currentSwipeDirection = direction
    animator = UIViewPropertyAnimator(duration: 0.8, curve: .easeIn, animations: {
        let transform = CGAffineTransform(translationX: direction == .right ? self.view.frame.width : -self.view.frame.width, y: 0)
        self.cardView.transform = CGAffineTransform(rotationAngle: direction == .right ? CGFloat(Double.pi/8) : -CGFloat(Double.pi/8)).concatenating(transform)
    })
}
```

```
private func refreshAnimator (direction : SwipeDirection) {
    currentSwipeDirection = direction
    animator.stopAnimation(true)
    animator = UIViewPropertyAnimator(duration: 0.8, curve: .easeIn, animations: {
        let transform = CGAffineTransform(translationX: direction == .right ? self.view.frame.width : -self.view.frame.width, y: 0)
        self.cardView.transform = CGAffineTransform(rotationAngle: direction == .right ? CGFloat(Double.pi/8) : -CGFloat(Double.pi/8)).concatenating(transform)
    })
}
```

```
func createNewCard () {
    cardView = UINib(nibName: "CardView", bundle: nil).instantiate(withOwner: nil, options: nil)[0] as! CardView
    cardView.frame = CGRect(x: 30, y: (view.frame.height - CGFloat(290))/2, width: view.frame.width - CGFloat(60), height: 290)
    view.addSubview(cardView)
    cardView.addGestureRecognizer(panRecognizer)
    cardView.isUserInteractionEnabled = true
    cardView.alpha = 0
    cardView.transform = CGAffineTransform(scaleX: 0.7, y: 0.7)
    UIView.animate(withDuration: 0.6) {
        self.cardView.alpha = 1.0
        self.cardView.transform = CGAffineTransform.identity
    }
}
```

II YouTube Pan Interaction

II View Controller - Swift

```
import UIKit
import UIKit.UIGestureRecognizerSubclass
import AVFoundation
private enum State {
    case closed, open
}
```

```
extension State {
```

```
    var opposite: State {  
        switch self {  
            case .open:  
                return .closed  
            case .closed:  
                return .open  
        }  
    }  
}
```

mp4 \Rightarrow make sure the target membership is right

```
class ViewController: UIViewController {
```

```
    private var animationProgress: CGFloat = 0  
    var transitionAnimator = UIViewControllerAnimated()  
    private var currentState: State = .closed  
    private lazy var panRecognizer: InstantPanGestureRecognizer = {  
        let recognizer = InstantPanGestureRecognizer()  
        recognizer.addTarget(self, action: #selector(handlePanGesture(recognizer:)))  
        return recognizer  
    }  
}
```

```
    lazy var thumbnailImageView: UIImageView = {  
        return UIImageView()  
    }  
}
```

```
    lazy var popupView: UIView = {  
        let _popupView = UIView()  
        _popupView.backgroundColor = UIColor.gray  
        return _popupView  
    }  
}
```

```
    var bottomConstraint = NSLayoutConstraint()  
    var popupOffset: CGFloat = 0  
    var viewHeight: CGFloat = 0  
}
```

```
    lazy var videoView: UIView = {  
        return UIView()  
    }  
}
```

```
    let videoURLString: String = Bundle.main.path(forResource: "clip", ofType: "mp4")!  
    var url: URL {  
        return URL(fileURLWithPath: videoURLString)  
    }  
}
```

```
    lazy var asset: AVURLAsset = {  
        var asset: AVURLAsset = AVURLAsset(url: url)  
        return asset  
    }  
}
```

```
    lazy var playerItem: AVPlayerItem = {  
        var playerItem: AVPlayerItem = AVPlayerItem(asset: self.asset)  
        return playerItem  
    }  
}
```

```
lazy var player: AVPlayer = {
    var player: AVPlayer = AVPlayer(playerItem: self.playerItem)
    player.actionAtItemEnd = .none
    return player
}

31)

lazy var playerLayer: AVPlayerLayer = {
    var playerLayer: AVPlayerLayer = AVPlayerLayer(player: self.player)
    return playerLayer
}

32)
```

```
override func viewDidLoad() {
    super.viewDidLoad()
    viewHeight = view.frame.size.height
    popupOffset = viewHeight - CGFloat(60)
    layout()
    popupView.addGestureRecognizer(panRecognizer)
}
```

```
func layout() {
    popupView.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(popupView)
    popupView.leadingAnchor.constraint(equalTo: view.leadingAnchor).isActive = true
    popupView.trailingAnchor.constraint(equalTo: view.trailingAnchor).isActive = true
    bottomConstraint = popupView.bottomAnchor.constraint(equalTo: view.bottomAnchor, constant: popupOffset)
    bottomConstraint.isActive = true
    popupView.heightAnchor.constraint(equalToConstant: viewHeight).isActive = true
    popupView.addSubview(videoView)
    videoView.translatesAutoresizingMaskIntoConstraints = false
    videoView.backgroundColor = UIColor.white
    videoView.leadingAnchor.constraint(equalTo: popupView.leadingAnchor, constant: 30).isActive = true
    videoView.trailingAnchor.constraint(equalTo: popupView.trailingAnchor, constant: -30).isActive = true
    videoView.topAnchor.constraint(equalTo: popupView.topAnchor, constant: 60).isActive = true
    NSLayoutConstraint(item: videoView, attribute: .height, relatedBy: .equal, toItem: videoView, attribute: .width, multiplier: 1, constant: 0).isActive = true
    popupView.addSubview(thumbnailImageView)
    thumbnailImageView.translatesAutoresizingMaskIntoConstraints = false
    thumbnailImageView.backgroundColor = UIColor.white
    thumbnailImageView.leadingAnchor.constraint(equalTo: popupView.leadingAnchor, constant: 15).isActive = true
    thumbnailImageView.topAnchor.constraint(equalTo: popupView.topAnchor, constant: 10).isActive = true
    thumbnailImageView.heightAnchor.constraint(equalToConstant: 40).isActive = true
    thumbnailImageView.widthAnchor.constraint(equalToConstant: 40).isActive = true
    thumbnailImageView.contentMode = .scaleAspectFit
    videoView.layer.insertSublayer(playerLayer, at: 0)
    thumbnailImageView.image = getThumbnailImage()
}
```

```
override func viewDidLoadSubviews() {
    super.viewDidLoadSubviews()
    playerLayer.frame = videoView.bounds
}
```

}

```
@objc func popupViewDammed(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
        case .began:
            if player.isPlaying {
                player.pause()
            }
        }
    thumbnailImageView.alpha = 0
    animateTransition(to: currentState.opposite, duration: 0.8)
    transitionAnimator.pauseAnimation()
    animationProgress = transitionAnimator.fractionComplete
}
case .changed:
    let translation = recognizer.translation(in: popupView)
    var fraction = -translation.y / popupOffset
    if currentState == .open & fraction != -1 {
        if transitionAnimator.isReversed & fraction != -1 {
            transitionAnimator.fractionComplete = fraction + animationProgress
        }
    }
}
```

case .ended

```
let yVelocity = recognizer.velocity(in: popupView).y
let shouldClose = yVelocity > 0
if yVelocity == 0 {
    transitionAnimator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
    break
}
switch currentState {
    case .open:
        if !shouldClose && !transitionAnimator.isReversed & transitionAnimator.isReversed =
            !transitionAnimator.isReversed {
            if shouldClose && transitionAnimator.isReversed & transitionAnimator.isReversed =
                !transitionAnimator.isReversed {
                    case .closed:
                        if shouldClose && !transitionAnimator.isReversed & transitionAnimator.isReversed =
                            !transitionAnimator.isReversed {
                                if !shouldClose && transitionAnimator.isReversed & transitionAnimator.isReversed =
                                    !transitionAnimator.isReversed {
                                        transitionAnimator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
}
}
}
}
}
default:
    ()
}
}
```

}

```

private func animateTransition(to state: State, duration: TimeInterval) {
    if transitionAnimator.isRunning { return }
    transitionAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1.0, animations: {
        switch state {
            case .open:
                self.bottomConstraint.constant = 0
            case .closed:
                self.bottomConstraint.constant = self.popupOffset
        }
        self.view.layoutIfNeeded()
    })
    transitionAnimator.addCompletion { [position] in
        switch position {
            case .start:
                self.currentState = state.opposite
            case .end:
                self.currentState = state
            case .current:
                break
            @unknown default:
                break
        }
    }
}

switch self.currentState {
    case .open:
        self.bottomConstraint.constant = 0
        self.player.play()
    case .closed:
        self.bottomConstraint.constant = self.popupOffset
        self.thumbnailImageView.transform = CGAffineTransform(scaleX: 1.2, y: 1.2)
        UIView.animate(withDuration: 0.2, animations: {
            self.thumbnailImageView.transform = CGAffineTransform.identity
            self.thumbnailImageView.alpha = 1.0
        })
}
}

// Switch
// completion
transitionAnimator.startAnimation()
}

```

```

func getThumbnailImage() -> UIImage? {
    let assetImageGenerator = AVAssetImageGenerator(asset: asset)
    assetImageGenerator.appliesPreferredTrackTransform = true
    let time = CMTimeMakeWithSeconds(Float64(15), preferredTimeScale: 600)
    do {
        let img = try assetImageGenerator.copyCGImage(at: time, actualTime: nil)
        let thumbnail = UIImage(cgImage: img)
        return thumbnail
    } catch {
        return nil
    }
}

```

extension AVPlayer

```
var !isPlaying : Bool {  
    return !(rate != 0 && error == nil())  
}
```

}

```
class InstantPanGestureRecognizer: UIPanGestureRecognizer {  
    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent) {  
        if !(self.state == UIGestureRecognizer.State.began) { return }  
        super.touchesBegan(touches, with: event)  
        self.state = UIGestureRecognizer.State.began  
    }  
}
```

UITabBar transition

```
import UIKit
```

```
let tabBarDelegate = TabBarDelegate()
```

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {  
    var window: UIWindow?  
  
    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:  
        [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  
        let tabController = UITabBarController()  
        let mainStoryboard = UIStoryboard(name: "Main", bundle: nil)  
        let firstVC = mainStoryboard.instantiateViewController(withIdentifier: "firstVC") as! ViewController  
        let secondVC = mainStoryboard.instantiateViewController(withIdentifier: "secondVC") as! secondViewController  
        firstVC.view.backgroundColor = UIColor.red  
        secondVC.view.backgroundColor = UIColor.orange  
        let vcData: [(UIViewController, UITabBarItem)] = [  
            (firstVC, UITabBarItem.init(tabBarSystemItem: UITabBarSystemItem.featured, tag: 1)),  
            (secondVC, UITabBarItem.init(tabBarSystemItem: UITabBarSystemItem.contacts, tag: 1))  
        ]  
  
        let vcs = vcData.map { (vc, tabBarItem) -> UINavigationController in  
            let nav = UINavigationController(rootViewController: vc)  
            nav.tabBarItem = tabBarItem  
            return nav  
        }  
        tabController.viewControllers = vcs  
        tabController.tabBar!.isTranslucent = false  
        tabController.delegate = tabBarDelegate  
        window?.rootViewController = tabController  
  
        return true  
    }  
}
```

//Tab Bar Delegate.swift

import UIKit

```
class TabBarDelegate : NSObject, UITabBarControllerDelegate {
    func tabBarController(_ tabBarController: UITabBarController, shouldSelect viewController: UIViewController) -> Bool {
        let selectedViewController = tabBarController.selectedViewController
        if selectedViewController == nil || viewController == selectedViewController {
            return false
        }

        guard let controllerIndex = tabBarController.viewControllers?.firstIndex(of: viewController) else { return true }

        guard let fromView = selectedViewController?.view else { return true }
        guard let toView = viewController.view else { return true }

        let viewSize: CGRect = fromView.frame
        let scrollRight: Bool = controllerIndex > tabBarController.selectedIndex
        fromView.superview?.addSubview(toView)

        let screenWidth: CGFloat = UIScreen.main.bounds.size.width
        toView.frame = CGRect(x: scrollRight ? screenWidth : -screenWidth, y: viewSize.origin.y, width: screenWidth, height: viewSize.size.height)

        UIView.animate(withDuration: 0.3, delay: 0, options: [.curveEaseInOut], animations: {
            fromView.frame = CGRect(x: scrollRight ? -screenWidth : screenWidth, y: viewSize.origin.y, width: screenWidth, height: viewSize.size.height)
            toView.frame = CGRect(x: 0, y: viewSize.origin.y, width: screenWidth, height: viewSize.size.height)
        }) { (success) in
            if (success) {
                fromView.removeFromSuperview()
                tabBarController.selectedIndex = controllerIndex
            }
        }
    }

    return false
}
```

//View Transitions

import UIKit

```
class ViewController : UIViewController {
    @IBOutlet weak var containerView: UIView!
    // @IBOutlet weak var redView: UIView!
    // @IBOutlet weak var orangeView: UIView!

    lazy var orangeView: UIView = {
        let orangeView = UIView()
        orangeView.backgroundColor = UIColor.orange
        return orangeView
    }
```

```
lazy var orangeView: UIView = {
    let orangeView = UIView()
    orangeView.backgroundColor = UIColor.orange
    return orangeView
}
```

```
lazy var redView: UIView = {
    let _redView = UIView()
    _redView.backgroundColor = UIColor.red
    return _redView
}

enum ViewType {
    case orange, red
}

var currentType: ViewType = .orange

override func viewDidLoad() {
    super.viewDidLoad()

    containerView.addSubview(redView)
    redView.frame.size = containerView.frame.size
    redView.frame.origin = CGPoint(x: 0, y: 0)
    containerView.addSubview(orangeView)
    orangeView.frame.size = containerView.frame.size
    orangeView.frame.origin = CGPoint(x: 0, y: 0)
}
```

```
@IBAction func transitionView(_ sender: Any) {
    switch currentType {
        case .orange:
            UIView.transition(with: containerView, duration: 1.0, options: [.transitionCurlUp], animations: {
                self.orangeView.removeFromSuperview()
                self.containerView.addSubview(self.redView)
            }) { (success) in
                self.currentType = .red
            }
        case .red:
            UIView.transition(with: containerView, duration: 1.0, options: [.transitionCurlUp], animations: {
                self.redView.removeFromSuperview()
                self.containerView.addSubview(self.orangeView)
            }) { (success) in
                self.currentType = .orange
            }
    }
}

//switch currentType {
//    case .orange:
//        UIView.transition(from: orangeView, to: redView, duration: 1.0, options: [.transitionFlipFromRight, .showHideTransitionViews])
//    } { (success) in
//        self.currentType = .red
//    }
//    case .red:
//        UIView.transition(from: redView, to: orangeView, duration: 1.0, options: [.transitionFlipFromLeft, .showHideTransitionViews])
//    } { (success) in
//        self.currentType = .orange
//    }
//}
```

