import UIKit

import CoreData

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {

var window: UIWindow?

func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

// Override point for customization after application launch.

SwiftyStoreKit.completeTransactions(atomically: true) { (purchases) in

for purchase in purchases{

switch purchase.transaction.transactionState {

case .purchased, .restored:

if purchase.needsFinishTransaction {

SwiftyStoreKit.finishTransaction(purchase.transaction)

}else{

// Do noting

}

case .failed, .purchasing, .deferred:

break

default:

break

}

}

}

// SVProgressHUD隐藏时间

SVProgressHUD.setMinimumDismissTimeInterval(1)

return true

}

// MARK: - Core Data stack

lazy var persistentContainer: NSPersistentContainer = {

/\*

The persistent container for the application. This implementation

creates and returns a container, having loaded the store for the

application to it. This property is optional since there are legitimate

error conditions that could cause the creation of the store to fail.

\*/

let container = NSPersistentContainer(name: "magnifier")

container.loadPersistentStores(completionHandler: { (storeDescription, error) in

if let error = error as NSError? {

// Replace this implementation with code to handle the error appropriately.

// fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.

/\*

Typical reasons for an error here include:

\* The parent directory does not exist, cannot be created, or disallows writing.

\* The persistent store is not accessible, due to permissions or data protection when the device is locked.

\* The device is out of space.

\* The store could not be migrated to the current model version.

Check the error message to determine what the actual problem was.

\*/

fatalError("Unresolved error \(error), \(error.userInfo)")

}

})

return container

}()

// MARK: - Core Data Saving support

func saveContext () {

let context = persistentContainer.viewContext

if context.hasChanges {

do {

try context.save()

} catch {

// Replace this implementation with code to handle the error appropriately.

// fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.

let nserror = error as NSError

fatalError("Unresolved error \(nserror), \(nserror.userInfo)")

}

}

}

}

import UIKit

@\_exported import CommonCrypto

@\_exported import SnapKit

@\_exported import SwiftyStoreKit

@\_exported import StoreKit

@\_exported import SVProgressHUD

class LBTools: NSObject {

}

let AppStoreAPPID = "382201985"// 假数据

let WebUrl\_Privacy = "https://www.jiaozhaobao.com/zoom/privacy.html" // 隐私协议

let WebUrl\_Terms = "https://www.jiaozhaobao.com/zoom/terms.html" // 服务条款

// 屏幕 宽度、高度

let SCREEN\_WIDTH = UIScreen.main.bounds.size.width

let SCREEN\_HEIGHT = UIScreen.main.bounds.size.height

// 判读是否为iPhone X及以上

let is\_iPhoneX = UIApplication.shared.statusBarFrame.size.height > 20 ? true : false

// 状态栏高度

let statusBarHeight = is\_iPhoneX ? 44 : 20

// 状态栏+导航栏高度

let statusBarAndNavigationBarHeight = is\_iPhoneX ? 88 : 64

// tabbar高度

let tabbarHeight = is\_iPhoneX ? 49+34 : 49

// 底部安全区域

let safeAreaBottom = is\_iPhoneX ? 34 : 0

// MARK: - 常用扩展

// color

extension UIColor {

static func hexString(\_ hex:String, a: Float) -> UIColor {

var cString:String = hex.trimmingCharacters(in: .whitespacesAndNewlines).uppercased()

if (cString.hasPrefix("#")) {

cString.remove(at: cString.startIndex)

}

if ((cString.count) != 6) {

return UIColor.gray

}

var rgbValue:UInt32 = 0

Scanner(string: cString).scanHexInt32(&rgbValue)

return UIColor(

red: CGFloat((rgbValue & 0xFF0000) >> 16) / 255.0,

green: CGFloat((rgbValue & 0x00FF00) >> 8) / 255.0,

blue: CGFloat(rgbValue & 0x0000FF) / 255.0,

alpha: CGFloat(a)

)

}

// UIColor.hexString("#2D73F1")

static func fromRGB(\_ rgbValue: UInt, a: Float) -> UIColor {

return UIColor(

red: CGFloat((rgbValue & 0xFF0000) >> 16) / 255.0,

green: CGFloat((rgbValue & 0x00FF00) >> 8) / 255.0,

blue: CGFloat(rgbValue & 0x0000FF) / 255.0,

alpha: CGFloat(a)

)

}

// UIColor.fromRGB(0x209624)

}

// Frame

extension UIView {

var width: CGFloat {

get { return self.frame.size.width }

set {

var frame = self.frame

frame.size.width = newValue

self.frame = frame

}

}

var height: CGFloat {

get { return self.frame.size.height }

set {

var frame = self.frame

frame.size.height = newValue

self.frame = frame

}

}

var size: CGSize {

get { return self.frame.size }

set {

var frame = self.frame

frame.size = newValue

self.frame = frame

}

}

var origin: CGPoint {

get { return self.frame.origin }

set {

var frame = self.frame

frame.origin = newValue

self.frame = frame

}

}

var x: CGFloat {

get { return self.frame.origin.x }

set {

var frame = self.frame

frame.origin.x = newValue

self.frame = frame

}

}

var y: CGFloat {

get { return self.frame.origin.y }

set {

var frame = self.frame

frame.origin.y = newValue

self.frame = frame

}

}

var centerX: CGFloat {

get { return self.center.x }

set {

self.center = CGPoint(x: newValue, y: self.center.y)

}

}

var centerY: CGFloat {

get { return self.center.y }

set {

self.center = CGPoint(x: self.center.x, y: newValue)

}

}

var top : CGFloat {

get { return self.frame.origin.y }

set {

var frame = self.frame

frame.origin.y = newValue

self.frame = frame

}

}

var bottom : CGFloat {

get { return frame.origin.y + frame.size.height }

set {

var frame = self.frame

frame.origin.y = newValue - self.frame.size.height

self.frame = frame

}

}

var right : CGFloat {

get { return self.frame.origin.x + self.frame.size.width }

set {

var frame = self.frame

frame.origin.x = newValue - self.frame.size.width

self.frame = frame

}

}

var left : CGFloat {

get { return self.frame.origin.x }

set {

var frame = self.frame

frame.origin.x = newValue

self.frame = frame

}

}

// 设置圆角、边框(方便在Storyboard中设置Layer层，圆角、边框、边框宽度和颜色等)

@IBInspectable var cornerRadius: CGFloat {

get {

return layer.cornerRadius

}

set {

layer.cornerRadius = newValue

layer.masksToBounds = newValue > 0

}

}

@IBInspectable var borderColor: UIColor {

get {

return UIColor.init(cgColor: layer.borderColor ??

UIColor.white.cgColor)

}

set {

layer.borderColor = newValue.cgColor

}

}

@IBInspectable var borderWidth: CGFloat {

get {

return layer.borderWidth

}

set {

layer.borderWidth = newValue

}

}

}

// 获取当前 毫秒级 时间戳 - 13位

extension Date {

var milliStamp : String {

let timeInterval: TimeInterval = self.timeIntervalSince1970

let millisecond = CLongLong(round(timeInterval\*1000))

return "\(millisecond)"

}

}

extension String{

//随机一个10到18位的字符串

static let random\_str\_characters = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

static func randomStr() -> String{

let len = arc4random() % 8 + 10

var ranStr = ""

for \_ in 0..<len {

let index = Int(arc4random\_uniform(UInt32(random\_str\_characters.count)))

ranStr.append(random\_str\_characters[random\_str\_characters.index(random\_str\_characters.startIndex, offsetBy: index)])

}

return ranStr

}

//sha1编码

func sha1() -> String {

let data = self.data(using: String.Encoding.utf8)!

var digest = [UInt8](repeating: 0, count: Int(CC\_SHA1\_DIGEST\_LENGTH))

CC\_SHA1([UInt8](data), CC\_LONG(data.count), &digest)

//无需base64输出,装换为16进制字符串输出

let output = NSMutableString(capacity: Int(CC\_SHA1\_DIGEST\_LENGTH))

for byte in digest {

output.appendFormat("%02x", byte)

}

return output as String

}

// MD5

func md5(strs:String) ->String{

let str = strs.cString(using: String.Encoding.utf8)

let strLen = CUnsignedInt(strs.lengthOfBytes(using: String.Encoding.utf8))

let digestLen = Int(CC\_MD5\_DIGEST\_LENGTH)

let result = UnsafeMutablePointer<CUnsignedChar>.allocate(capacity: digestLen)

CC\_MD5(str!, strLen, result)

let hash = NSMutableString()

for i in 0 ..< digestLen {

hash.appendFormat("%02x", result[i])

}

return String(format: hash as String)

}

}

/// JSONString转换为字典

/// - Parameter jsonString: JSONString

/// - Returns: NSDictionary

public func getDictionaryFromJSONString(jsonString:String) ->NSDictionary{

let jsonData:Data = jsonString.data(using: .utf8)!

let dict = try? JSONSerialization.jsonObject(with: jsonData, options: .mutableContainers)

if dict != nil {

return dict as! NSDictionary

}

return NSDictionary()

}

/// 字典转换为JSONString

/// - Parameter dictionary: NSDictionary

/// - Returns: JSONString

func getJSONStringFromDictionary(dictionary:NSDictionary) -> String {

if (!JSONSerialization.isValidJSONObject(dictionary)) {

print("无法解析出JSONString")

return ""

}

let data : NSData! = try? JSONSerialization.data(withJSONObject: dictionary, options: []) as NSData?

let JSONString = NSString(data:data as Data,encoding: String.Encoding.utf8.rawValue)

return JSONString! as String

}

public extension UIDevice {

static let modelName: String = {

var systemInfo = utsname()

uname(&systemInfo)

let machineMirror = Mirror(reflecting: systemInfo.machine)

let identifier = machineMirror.children.reduce("") { identifier, element in

guard let value = element.value as? Int8, value != 0 else { return identifier }

return identifier + String(UnicodeScalar(UInt8(value)))

}

func mapToDevice(identifier: String) -> String { // swiftlint:disable:this cyclomatic\_complexity

#if os(iOS)

switch identifier {

case "iPod5,1": return "iPod touch (5th generation)"

case "iPod7,1": return "iPod touch (6th generation)"

case "iPod9,1": return "iPod touch (7th generation)"

case "iPhone3,1", "iPhone3,2", "iPhone3,3": return "iPhone 4"

case "iPhone4,1": return "iPhone 4s"

case "iPhone5,1", "iPhone5,2": return "iPhone 5"

case "iPhone5,3", "iPhone5,4": return "iPhone 5c"

case "iPhone6,1", "iPhone6,2": return "iPhone 5s"

case "iPhone7,2": return "iPhone 6"

case "iPhone7,1": return "iPhone 6 Plus"

case "iPhone8,1": return "iPhone 6s"

case "iPhone8,2": return "iPhone 6s Plus"

case "iPhone9,1", "iPhone9,3": return "iPhone 7"

case "iPhone9,2", "iPhone9,4": return "iPhone 7 Plus"

case "iPhone8,4": return "iPhone SE"

case "iPhone10,1", "iPhone10,4": return "iPhone 8"

case "iPhone10,2", "iPhone10,5": return "iPhone 8 Plus"

case "iPhone10,3", "iPhone10,6": return "iPhone X"

case "iPhone11,2": return "iPhone XS"

case "iPhone11,4", "iPhone11,6": return "iPhone XS Max"

case "iPhone11,8": return "iPhone XR"

case "iPhone12,1": return "iPhone 11"

case "iPhone12,3": return "iPhone 11 Pro"

case "iPhone12,5": return "iPhone 11 Pro Max"

case "iPad2,1", "iPad2,2", "iPad2,3", "iPad2,4":return "iPad 2"

case "iPad3,1", "iPad3,2", "iPad3,3": return "iPad (3rd generation)"

case "iPad3,4", "iPad3,5", "iPad3,6": return "iPad (4th generation)"

case "iPad6,11", "iPad6,12": return "iPad (5th generation)"

case "iPad7,5", "iPad7,6": return "iPad (6th generation)"

case "iPad7,11", "iPad7,12": return "iPad (7th generation)"

case "iPad4,1", "iPad4,2", "iPad4,3": return "iPad Air"

case "iPad5,3", "iPad5,4": return "iPad Air 2"

case "iPad11,4", "iPad11,5": return "iPad Air (3rd generation)"

case "iPad2,5", "iPad2,6", "iPad2,7": return "iPad mini"

case "iPad4,4", "iPad4,5", "iPad4,6": return "iPad mini 2"

case "iPad4,7", "iPad4,8", "iPad4,9": return "iPad mini 3"

case "iPad5,1", "iPad5,2": return "iPad mini 4"

case "iPad11,1", "iPad11,2": return "iPad mini (5th generation)"

case "iPad6,3", "iPad6,4": return "iPad Pro (9.7-inch)"

case "iPad6,7", "iPad6,8": return "iPad Pro (12.9-inch)"

case "iPad7,1", "iPad7,2": return "iPad Pro (12.9-inch) (2nd generation)"

case "iPad7,3", "iPad7,4": return "iPad Pro (10.5-inch)"

case "iPad8,1", "iPad8,2", "iPad8,3", "iPad8,4":return "iPad Pro (11-inch)"

case "iPad8,5", "iPad8,6", "iPad8,7", "iPad8,8":return "iPad Pro (12.9-inch) (3rd generation)"

case "AppleTV5,3": return "Apple TV"

case "AppleTV6,2": return "Apple TV 4K"

case "AudioAccessory1,1": return "HomePod"

case "i386", "x86\_64": return "Simulator \(mapToDevice(identifier: ProcessInfo().environment["SIMULATOR\_MODEL\_IDENTIFIER"] ?? "iOS"))"

default: return identifier

}

#elseif os(tvOS)

switch identifier {

case "AppleTV5,3": return "Apple TV 4"

case "AppleTV6,2": return "Apple TV 4K"

case "i386", "x86\_64": return "Simulator \(mapToDevice(identifier: ProcessInfo().environment["SIMULATOR\_MODEL\_IDENTIFIER"] ?? "tvOS"))"

default: return identifier

}

#endif

}

return mapToDevice(identifier: identifier)

}()

}

extension UIImage{

class func Create(size:CGSize,color : UIColor,alpha:CGFloat? = 1) -> UIImage{

UIGraphicsBeginImageContext(size)

color.setFill()

let bounds = CGRect.init(x: 0, y: 0, width: size.width, height: size.height)

UIRectFill(bounds)

UIImage().draw(in: bounds, blendMode: CGBlendMode.destinationIn, alpha: alpha ?? 1)

let tintedImage = UIGraphicsGetImageFromCurrentImageContext()

UIGraphicsEndImageContext()

return tintedImage!

}

}

import UIKit

//import SwiftyStoreKit

//import StoreKit

//import SVProgressHUD

class LBIAPTool: NSObject {

static let sharedInstance: LBIAPTool = {

let instance = LBIAPTool()

return instance

}()

// 通过product id 购买商品

func payWithProduct(productID: String,payResult: ((Bool)->())? = nil) {

SVProgressHUD.show()

SwiftyStoreKit.purchaseProduct(productID, quantity: 1, atomically: false) { result in

switch result {

case .success(let product):

// fetch content from your server, then:

// 服务器验证

// self.verifyPurchase(forceupdate:true,result: { result in

// payResult?(result)

//// if result{

//// SVProgressHUD.showSuccess(withStatus: "购买成功")

//// }else{

//// SVProgressHUD.showError(withStatus: "购买失败")

//// }

// })

// 本地验证

self.verifyReceipt(service: .production)

if product.needsFinishTransaction {

SwiftyStoreKit.finishTransaction(product.transaction)

}

print("Purchase Success: \(product.productId)")

case .error(let error):

SVProgressHUD.dismiss()

switch error.code {

case .unknown: print("Unknown error. Please contact support")

case .clientInvalid: print("Not allowed to make the payment")

case .paymentCancelled: break

case .paymentInvalid: print("The purchase identifier was invalid")

case .paymentNotAllowed: print("The device is not allowed to make the payment")

case .storeProductNotAvailable: print("The product is not available in the current storefront")

case .cloudServicePermissionDenied: print("Access to cloud service information is not allowed")

case .cloudServiceNetworkConnectionFailed: print("Could not connect to the network")

case .cloudServiceRevoked: print("User has revoked permission to use this cloud service")

default: print((error as NSError).localizedDescription)

}

}

}

}

// 恢复购买

func restorePurchases(success:((\_ resulet:Bool)->())?) {

SVProgressHUD.show()

let signal = DispatchSemaphore(value: 0)

var isSuccess = false

DispatchQueue.global().async {

SwiftyStoreKit.restorePurchases(atomically: true) { results in

SVProgressHUD.dismiss()

if results.restoreFailedPurchases.count > 0 {

SVProgressHUD.showError(withStatus: "恢复购买失败")

print("Restore Failed: \(results.restoreFailedPurchases)")

isSuccess = false

signal.signal()

}else if results.restoredPurchases.count > 0 {

print("Restore Success: \(results.restoredPurchases)")

// 本地验证

self.verifyReceipt(service: .production)

// 服务器验证

// self.verifyPurchase(isRest: true) { res in

// isSuccess = res

// signal.signal()

// if res{

// SVProgressHUD.showSuccess(withStatus: "恢复购买成功")

// }else{

// SVProgressHUD.showError(withStatus: "恢复购买失败")

// }

//

// }

}else {

SVProgressHUD.showError(withStatus: "没有可恢复的")

print("Nothing to Restore")

isSuccess = false

signal.signal()

}

}

signal.wait()

if success != nil{

success!(isSuccess)

}

}

}

//服务器验证内购

func verifyPurchase(forceupdate:Bool?=false,isRest:Bool? = false,result:((Bool)->())? = nil) {

let receipUrl = Bundle.main.appStoreReceiptURL

let receiptData = try? Data(contentsOf: receipUrl!)

let receiptString = receiptData?.base64EncodedString(options: .endLineWithLineFeed) ?? ""

//服务器验证请求

// NetworkingTool.newVerifyReceipt(receiptStr: receiptString, forceupdate: forceupdate, success: { response in

// weak var weakSelf = self

// if forceupdate == false && response.code == 403{

// result?(false)

// SVProgressHUD.dismiss()

// }else if response.code == 200{

// print("\(response.message)---\(response.expires)")

// if response.message == "ok"{

// UserDefaults.standard.set(response.product\_id, forKey: "product\_id")

// weakSelf?.updateExpireTime(time: Int(response.expires) ?? 0)

// if weakSelf?.getCurVipStatus() ?? false {

// result?(true)

// if forceupdate ?? false && isRest == false{

// SVProgressHUD.showSuccess(withStatus: NSLocalizedString("purchase\_success", comment: ""))

// }

// }else{

// result?(false)

// if forceupdate ?? false && isRest == false{

// SVProgressHUD.showError(withStatus: NSLocalizedString("purchase\_failed", comment: ""))

// }

// }

// }else{

// result?(false)

// if forceupdate ?? false && isRest == false{

// SVProgressHUD.showError(withStatus: NSLocalizedString("purchase\_failed", comment: ""))

// }

// }

// }else{

// result?(false)

// SVProgressHUD.showError(withStatus: NSLocalizedString("purchase\_failed", comment: ""))

// print("服务器错误")

// }

//

// }) { error in

// SVProgressHUD.dismiss()

// print(error)

// }

}

//内购本地验证

func verifyReceipt(service:AppleReceiptValidator.VerifyReceiptURLType,res:((Bool)->())? = nil) {

let receiptValidator = AppleReceiptValidator(service: service, sharedSecret: "23bbbc327dd54f03aec4f14acb58a331")

SwiftyStoreKit.verifyReceipt(using: receiptValidator) { (result) in

switch result {

case .success (let receipt):

SVProgressHUD.dismiss()

let status: Int = receipt["status"] as! Int

if status == 21007 {// 沙盒测试模式

// sandbox验证

self.verifyReceipt(service: .sandbox)

}

print("receipt：\(receipt)")

SVProgressHUD.showSuccess(withStatus: "购买成功")

let dic = receipt["receipt"]

let dicInApp = dic?["in\_app"] as! Array<Dictionary<String,Any>>

print(dicInApp as Any)

// 计算过期时间

var expireTime = 0;

for iapItem in dicInApp{

let iapTime = Int(iapItem["expires\_date\_ms"] as? String ?? "0") ?? 0

print(expireTime)

if expireTime < iapTime {

expireTime = iapTime

}

}

print("lastTime:\(expireTime)")

// 更新过期时间

weak var weakSelf = self

weakSelf?.updateExpireTime(time: expireTime)

break

case .error(let error):

SVProgressHUD.dismiss()

print("error：\(error)")

break

}

}

}

// 更新到期时间

func updateExpireTime(time:Int){

let defaults = UserDefaults.standard

defaults.set(time, forKey: "EXPIRE\_TIME")

}

// 获取当前VIP状态

func getCurrentVIPStatus() -> Bool{

let defaults = UserDefaults.standard

let expTime = defaults.integer(forKey: "EXPIRE\_TIME")

let now = self.getNowTimeStampString()

if expTime > Int(now) ?? 0 {// 是VIP

return true

}else{

// return false

return true // 假数据，先去掉内购

}

}

//获取当前时间戳

func getNowTimeStampString()->String{

let date = Date()

let timeInterval:TimeInterval = TimeInterval(date.timeIntervalSince1970)

let millisecond = CLongLong(round(timeInterval \* 1000))

return "\(millisecond)"

}

}

import UIKit

import WebKit

class LBWebViewController: UIViewController {

var url : String = ""

@objc var webView = WKWebView()

lazy var myProgressView : UIProgressView = {

let ProgressView = UIProgressView.init(frame: CGRect(x: 0, y: 0, width: SCREEN\_WIDTH, height: 1))

ProgressView.tintColor = UIColor.blue

ProgressView.trackTintColor = UIColor.white

return ProgressView

}()

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

self.navigationController?.navigationBar.isTranslucent = false

}

override func viewWillDisappear(\_ animated: Bool) {

super.viewWillDisappear(animated)

self.navigationController?.navigationBar.isTranslucent = true

}

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.webView = WKWebView.init()

self.view.addSubview(self.webView)

self.webView.snp.makeConstraints { (make) in

make.left.equalToSuperview().offset(0)

make.right.equalToSuperview().offset(0)

make.top.equalToSuperview().offset(0)

make.bottom.equalToSuperview().offset(0)

}

let request = URLRequest.init(url: URL.init(string: self.url)!)

self.webView.load(request)

// 进度条

self.view.addSubview(self.myProgressView)

self.webView.addObserver(self, forKeyPath: "estimatedProgress", options: NSKeyValueObservingOptions.new, context: nil)

}

deinit {

self.webView.removeObserver(self, forKeyPath: "estimatedProgress")

}

override func observeValue(forKeyPath keyPath: String?, of object: Any?, change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {

if keyPath == "estimatedProgress" {

self.myProgressView.progress = Float(webView.estimatedProgress)

if (self.myProgressView.progress >= 1.0) {

let deadline = DispatchTime.now() + 0.3

DispatchQueue.global().asyncAfter(deadline: deadline) {

DispatchQueue.main.async {

self.myProgressView.progress = 0;

}

}

}

}else{

}

}

/\*

// 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.

}

\*/

}

import UIKit

import AVFoundation

import Photos

//import AudioToolbox

enum FilterMode {

case No

case Yellow\_Black

case Yellow\_Blue

case Reverse

case Black\_White

}

class LBCameraViewController: LBBaseViewController {

var device : AVCaptureDevice? // 获取设备：如摄像头

let session = AVCaptureSession() // 会话，协调着input到output的数据传输，input和output的桥梁

var videoDataOutput : AVCaptureVideoDataOutput? // 输出

var takeImage : UIImage? // 实时获取到的照片

var previewLayer : CALayer = CALayer.init() // 实时显示预览图片

var currentFilterMode : FilterMode = .No

// 焦距拖动图片

lazy var focalLengthImageView: UIImageView = {

let imageView = UIImageView()

imageView.image = UIImage.init(named: "focal\_length.png")

imageView.isUserInteractionEnabled = true

imageView.isHidden = true

// imageView.backgroundColor = .red

return imageView

}()

// 焦距拖动手势

lazy var panView: UIView = {

let view = UIView()

view.isUserInteractionEnabled = true

view.isHidden = true

// view.backgroundColor = .yellow

// view.alpha = 0.3

// 添加拖动手势

let panGesture = UIPanGestureRecognizer.init(target: self, action: #selector(focalLengthImageViewPan(sender:)))

view.addGestureRecognizer(panGesture)

return view

}()

// 焦距显示倍率

lazy var defaultFocalLengthLabel: UILabel = {

let label = UILabel()

label.textAlignment = .center

label.font = UIFont.init(name: "Helvetica-Bold", size: 14)

label.textColor = UIColor.white

label.text = "0.0x"

return label

}()

var lengthStr = "0.0x" // 记录旧值，用来实现值发生改变震动效果

// 焦距记录现在旋转的角度

var saveAngle : Float = 0.0

// 相机点击手势聚焦view

lazy var focusView: UIView = {

let view = UIView()

view.backgroundColor = UIColor.clear

view.layer.borderColor = UIColor.hexString("#FFE100", a: 1.0).cgColor

view.layer.borderWidth = 1

view.isHidden = true

return view

}()

// UI

@IBOutlet weak var defaultStackView: UIStackView!

@IBOutlet weak var defaultFilterButton: UIButton!

@IBOutlet weak var defaultLightButton: UIButton!

@IBOutlet weak var defaultFocalLengthButton: UIButton!

@IBOutlet weak var lightFocalLengthButton: UIButton!

@IBOutlet weak var filterFocalLengthButton: UIButton!

@IBOutlet weak var lockStackView: UIStackView!

@IBOutlet weak var lightView: UIView!

@IBOutlet weak var lightSliderView: SectionedSlider!

@IBOutlet weak var filterStackView: UIStackView!

@IBOutlet weak var vipImageView: UIImageView!

@IBOutlet weak var filterYellowBlackVipImageView: UIImageView!

@IBOutlet weak var filterYellowBlueVipImageView: UIImageView!

@IBOutlet weak var filterReverseVipImageView: UIImageView!

// 隐私政策弹窗

func privacyView() -> Void {

let userDefault = UserDefaults.standard

if (userDefault.object(forKey: "Privacy") != nil) {

// 已查看过

}else{

// 未查看

let vc = LBPrivacyViewController.init()

vc.acceptBlock = {

userDefault.setValue("YES", forKey: "Privacy")

userDefault.synchronize()

}

vc.modalPresentationStyle = .fullScreen

self.present(vc, animated: false) {

}

}

}

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

if session.isRunning == false {

session.startRunning()

}

self.navigationController?.setNavigationBarHidden(true, animated: animated)

// 隐私政策弹窗

self.privacyView()

}

override func viewWillDisappear(\_ animated: Bool) {

super.viewWillDisappear(animated)

self.navigationController?.setNavigationBarHidden(false, animated: animated)

}

override func viewDidDisappear(\_ animated: Bool) {

super.viewDidDisappear(animated)

if session.isRunning {

session.stopRunning()

}

}

override func viewDidLayoutSubviews() {

super.viewDidLayoutSubviews()

// print("66666")

// 焦距距离调节控件

let y = self.defaultStackView.frame.origin.y + self.defaultStackView.frame.size.height - 48// 计算Y坐标

let radius = SCREEN\_HEIGHT - self.defaultStackView.bottom + 48// 计算半径

// 焦距尺度显示图片

if self.view.subviews.contains(self.focalLengthImageView) == false {

self.focalLengthImageView.frame = CGRect(x: SCREEN\_WIDTH - radius, y: CGFloat(y), width: radius \* 2, height: radius \* 2)

self.view.insertSubview(self.focalLengthImageView, belowSubview: self.defaultStackView)// 之下

}

// 焦距手势控件

if self.view.subviews.contains(self.panView) == false {

self.panView.frame = CGRect(x: SCREEN\_WIDTH - radius, y: CGFloat(y), width: radius, height: radius)

self.view.insertSubview(self.panView, aboveSubview: self.focalLengthImageView)// 之上

}

// 焦距倍率控件

if self.view.subviews.contains(self.defaultFocalLengthLabel) == false {

self.defaultFocalLengthLabel.frame = CGRect(x: self.defaultStackView.x, y: self.defaultStackView.bottom-40, width: 40, height: 40)

self.view.addSubview(self.defaultFocalLengthLabel)

}

}

override func viewDidLoad() {

super.viewDidLoad()

// 灯光亮度调节控件

self.lightSliderView.delegate = self

// 相机权限

let captureStatus = AVCaptureDevice.authorizationStatus(for: .video)

switch captureStatus {

case .authorized: //允许状态

self.loadCamera()

case .denied: //禁止状态

self.showAlert(title: "Please open camera permissions", message: "Settings-Privacy-Camera")

case .notDetermined: //用户从未对相机授权做操作,第一次开启时

self.loadCamera()

case .restricted: // 受限制的

self.showAlert(title: "Please open camera permissions", message: "Settings-Privacy-Camera")

default:

break

}

// 点击屏幕聚焦

self.focusView.frame = CGRect(x: 0, y: 0, width: 80, height: 80)

self.view.addSubview(self.focusView)

}

func loadCamera() -> Void {

// SessionPreset,用于设置output输出流的画面质量

session.sessionPreset = .inputPriority

// 获取输入设备,builtInWideAngleCamera是通用相机,AVMediaType.video代表视频媒体,back表示前置摄像头,如果需要后置摄像头修改为front

device = AVCaptureDevice.default(for: .video)

// 设置输入流

if let dev = device {

let input = try? AVCaptureDeviceInput(device: dev)

if let inputt = input{

//连接输入与会话

if session.canAddInput(inputt)==true{

session.addInput(inputt)

}

}

}

// 设置视频输出流

videoDataOutput = AVCaptureVideoDataOutput.init()

videoDataOutput!.videoSettings = [kCVPixelBufferPixelFormatTypeKey as NSString as String : kCVPixelFormatType\_32BGRA]

videoDataOutput!.alwaysDiscardsLateVideoFrames = true // 丢弃延迟的帧

let queue = DispatchQueue(label: "VideoDataOutputQueue")

videoDataOutput!.setSampleBufferDelegate(self, queue: queue)

if session.canAddOutput(videoDataOutput! as AVCaptureOutput) == true {

session.addOutput(videoDataOutput! as AVCaptureOutput)

}

// 图像预览层，实时显示捕获的图像

previewLayer.frame = CGRect(x: 0, y: 0, width: SCREEN\_WIDTH, height: SCREEN\_HEIGHT)

self.view.layer.insertSublayer(previewLayer, at: 0)

session.startRunning()

// 相机点击对焦手势

let tap = UITapGestureRecognizer.init(target: self, action: #selector(tapScreen(sender:)))

self.view.addGestureRecognizer(tap)

}

@objc func tapScreen(sender:UITapGestureRecognizer) -> Void {

let point = sender.location(in: sender.view)

let size = self.view.bounds.size

let focusPoint = CGPoint(x: point.y/size.height, y: 1-point.x/size.width)

do {

try device?.lockForConfiguration()

// 对焦模式和对焦点

if ((self.device?.isFocusModeSupported(.autoFocus)) != nil) {

self.device?.focusPointOfInterest = focusPoint

self.device?.focusMode = .autoFocus

}

device?.unlockForConfiguration()

} catch {

print(error)

}

// 设置对焦动画

self.focusView.center = point

self.focusView.isHidden = false

UIView.animate(withDuration: 0.3) {

self.focusView.transform = CGAffineTransform(scaleX: 1.25, y: 1.25)

} completion: { (finished) in

UIView.animate(withDuration: 0.5) {

self.focusView.transform = CGAffineTransform.identity

} completion: { (finished) in

self.focusView.isHidden = true

}

}

}

// MARK: - Action

// 设置

@IBAction func settingButtonAction(\_ sender: UIButton) {

}

// 滤镜

@IBAction func filterButtonAction(\_ sender: UIButton) {

self.defaultStackView.isHidden = true

self.filterStackView.isHidden = false

self.filterFocalLengthButton.isSelected = self.defaultFocalLengthButton.isSelected

// 判断是否已购买

if (LBIAPTool.sharedInstance.getCurrentVIPStatus()) {

// 已充值

self.filterYellowBlackVipImageView.isHidden = true

self.filterYellowBlueVipImageView.isHidden = true

self.filterReverseVipImageView.isHidden = true

}else{

// 未充值,限制免费账号

self.filterYellowBlackVipImageView.isHidden = false

self.filterYellowBlueVipImageView.isHidden = false

self.filterReverseVipImageView.isHidden = false

}

}

// 灯光

@IBAction func lightButtonAction(\_ sender: UIButton) {

self.defaultStackView.isHidden = true

self.lightView.isHidden = false

self.lightFocalLengthButton.isSelected = self.defaultFocalLengthButton.isSelected

}

@IBAction func lightCloseButtonAction(\_ sender: UIButton) {

self.defaultStackView.isHidden = false

self.lightView.isHidden = true

self.defaultFocalLengthButton.isSelected = self.lightFocalLengthButton.isSelected

}

// 锁

@IBAction func lockButtonAction(\_ sender: UIButton) {

switch sender.tag {

case 41:

self.defaultStackView.isHidden = true

self.lockStackView.isHidden = false

self.session.stopRunning()

// 焦距逻辑----

self.defaultFocalLengthLabel.isHidden = true

// 隐藏焦距滑动圈

self.defaultFocalLengthButton.isSelected = false

self.focalLengthImageView.isHidden = true

self.panView.isHidden = true

self.defaultFocalLengthLabel.textColor = UIColor.white

case 42:

self.defaultStackView.isHidden = false

self.lockStackView.isHidden = true

session.startRunning()

// 焦距逻辑----

self.defaultFocalLengthLabel.isHidden = false

default:

break

}

}

// 锁状态下保存照片到相册

@IBAction func lock\_savePhotoButtonAction(\_ sender: UIButton) {

self.saveImageToPhotoLibrary(image: takeImage)

}

// 拍照

@IBAction func takePhotoButtonAction(\_ sender: UIButton) {

self.saveImageToPhotoLibrary(image: takeImage)

}

// 焦距

@IBAction func focalLengthButtonAction(\_ sender: UIButton) {

if self.vipImageView.isHidden == false {

// 跳转内购页面

let vc = LBBuyVIPViewController()

vc.BuyButtonSuccessBlock = {

self.vipImageView.isHidden = true

self.filterYellowBlackVipImageView.isHidden = true

self.filterYellowBlueVipImageView.isHidden = true

self.filterReverseVipImageView.isHidden = true

}

self.present(vc, animated: true) {

}

}else{

// 正常逻辑

sender.isSelected = !sender.isSelected

if sender.isSelected {

self.focalLengthImageView.isHidden = false

self.panView.isHidden = false

self.defaultFocalLengthLabel.textColor = UIColor.hexString("#222222", a: 1.0)

}else{

self.focalLengthImageView.isHidden = true

self.panView.isHidden = true

self.defaultFocalLengthLabel.textColor = UIColor.white

}

}

}

// 黄黑滤镜

@IBAction func filterYellowWhiteButtonAction(\_ sender: UIButton) {

// 判断是否已购买

if (LBIAPTool.sharedInstance.getCurrentVIPStatus()) {

// 已充值

self.defaultStackView.isHidden = false

self.filterStackView.isHidden = true

self.defaultFocalLengthButton.isSelected = self.filterFocalLengthButton.isSelected

self.defaultFilterButton.setImage(UIImage.init(named: "filter\_yellow\_white"), for: .normal)

// 开启滤镜 - 黄黑

self.currentFilterMode = .Yellow\_Black

}else{

// 未充值,跳转内购界面

let vc = LBBuyVIPViewController()

vc.BuyButtonSuccessBlock = {

self.vipImageView.isHidden = true

self.filterYellowBlackVipImageView.isHidden = true

self.filterYellowBlueVipImageView.isHidden = true

self.filterReverseVipImageView.isHidden = true

}

self.present(vc, animated: true) {

}

}

}

// 黄蓝滤镜

@IBAction func filterYellowBlueButtonAction(\_ sender: UIButton) {

if (LBIAPTool.sharedInstance.getCurrentVIPStatus()) {

// 已充值

self.defaultStackView.isHidden = false

self.filterStackView.isHidden = true

self.defaultFocalLengthButton.isSelected = self.filterFocalLengthButton.isSelected

self.defaultFilterButton.setImage(UIImage.init(named: "filter\_yellow\_blue"), for: .normal)

// 开启滤镜 - 黄蓝

self.currentFilterMode = .Yellow\_Blue

}else{

// 未充值,跳转内购界面

let vc = LBBuyVIPViewController()

vc.BuyButtonSuccessBlock = {

self.vipImageView.isHidden = true

self.filterYellowBlackVipImageView.isHidden = true

self.filterYellowBlueVipImageView.isHidden = true

self.filterReverseVipImageView.isHidden = true

}

self.present(vc, animated: true) {

}

}

}

// 反转滤镜

@IBAction func filterReverseButtonAction(\_ sender: UIButton) {

if (LBIAPTool.sharedInstance.getCurrentVIPStatus()) {

// 已充值

self.defaultStackView.isHidden = false

self.filterStackView.isHidden = true

self.defaultFocalLengthButton.isSelected = self.filterFocalLengthButton.isSelected

self.defaultFilterButton.setImage(UIImage.init(named: "filter\_reverse"), for: .normal)

// 开启滤镜 - 反转

self.currentFilterMode = .Reverse

}else{

// 未充值,跳转内购界面

let vc = LBBuyVIPViewController()

vc.BuyButtonSuccessBlock = {

self.vipImageView.isHidden = true

self.filterYellowBlackVipImageView.isHidden = true

self.filterYellowBlueVipImageView.isHidden = true

self.filterReverseVipImageView.isHidden = true

}

self.present(vc, animated: true) {

}

}

}

// 黑白滤镜

@IBAction func filterBlackWhiteButtonAction(\_ sender: UIButton) {

self.defaultStackView.isHidden = false

self.filterStackView.isHidden = true

self.defaultFocalLengthButton.isSelected = self.filterFocalLengthButton.isSelected

self.defaultFilterButton.setImage(UIImage.init(named: "filter\_black\_white"), for: .normal)

// 开启滤镜 - 黑白

self.currentFilterMode = .Black\_White

// 隐藏vip限制图标

self.filterYellowBlackVipImageView.isHidden = true

self.filterYellowBlueVipImageView.isHidden = true

self.filterReverseVipImageView.isHidden = true

}

// 关闭滤镜

@IBAction func filterCloseButtonAction(\_ sender: UIButton) {

self.defaultStackView.isHidden = false

self.filterStackView.isHidden = true

self.defaultFocalLengthButton.isSelected = self.filterFocalLengthButton.isSelected

self.defaultFilterButton.setImage(UIImage.init(named: "filter\_default"), for: .normal)

// 关闭滤镜

self.currentFilterMode = .No

// 隐藏vip限制图标

self.filterYellowBlackVipImageView.isHidden = true

self.filterYellowBlueVipImageView.isHidden = true

self.filterReverseVipImageView.isHidden = true

}

// 拖动手势监听

@objc func focalLengthImageViewPan(sender:UIPanGestureRecognizer) -> Void {

if (sender.state == .changed) {

self.commitTranslation(translation: sender.translation(in: panView))

}else{

// Do noting

}

}

// 判断拖动手势方向

func commitTranslation(translation:CGPoint) -> Void {

// let absX = abs(translation.x)

// let absY = abs(translation.y)

// 降低手势响应灵敏度(默认是1像素旋转一度，现在是10像素旋转一度)

var absX = abs(translation.x)

var absY = abs(translation.y)

absX = absX / 10

absY = absY / 10

// print(absX, absY)

if absX > absY {

if translation.x < 0{// 向左滑动

saveAngle -= Float(absX)

}else{// 向右滑动

saveAngle += Float(absX)

}

}else if absY > absX{

if translation.y < 0 {// 向上滑动

saveAngle += Float(absY)

}else{// 向下滑动

saveAngle -= Float(absY)

}

}else{

// Do noting

}

if saveAngle < 0{

saveAngle = 0 // 起点

}else if saveAngle > 90{

saveAngle = 90 // 终点

}else{

// 每格旋转角度等于 90/20 = 4.5度 ，取余数，旋转

saveAngle = (saveAngle.truncatingRemainder(dividingBy: 4.5)==0) ? saveAngle : (saveAngle - saveAngle.truncatingRemainder(dividingBy: 4.5))

}

print(saveAngle)

if (LBIAPTool.sharedInstance.getCurrentVIPStatus()) {

// 已充值

}else{

// 未充值,限制免费账号

if saveAngle > 45 {

saveAngle = 45

self.vipImageView.isHidden = false

}else{

self.vipImageView.isHidden = true

}

}

// 拿到一个度数 度数/90 = x/ 0.5

let x = CGFloat(saveAngle \* 0.5 / 90.0)

// 旋转

UIView.animate(withDuration: 0.05) {

self.focalLengthImageView.transform = CGAffineTransform.identity

self.focalLengthImageView.transform = CGAffineTransform(rotationAngle: .pi \* x)

}

let currentZoomFactor = self.saveAngle/4.5 // 当前倍数

let each = CGFloat(self.maxZoomFactor()/20) // 每格的放大参数

self.defaultFocalLengthLabel.text = "\(currentZoomFactor)x"

// 如果值发生变化，就震动一次

if self.defaultFocalLengthLabel.text != lengthStr {

// 震动

let generator = UIImpactFeedbackGenerator(style: .medium)

generator.impactOccurred()

lengthStr = self.defaultFocalLengthLabel.text!

}else{

// Do noting

}

// 放大倍数 = 当前倍数 \* 每格的放大参数

var n = CGFloat(currentZoomFactor) \* each

if n < minZoomFactor() {

n = minZoomFactor()

}else if n > maxZoomFactor(){

n = maxZoomFactor()

}else{

// Do noting

}

do {

try device?.lockForConfiguration()

// 改变设备当前远近焦距

self.device?.videoZoomFactor = n

device?.unlockForConfiguration()

} catch {

print(error)

}

}

// 最小缩放值

func minZoomFactor() -> CGFloat {

var minZoomFactor = CGFloat(1.0)

if #available(iOS 11.0, \*) {

minZoomFactor = self.device!.minAvailableVideoZoomFactor

} else {

// Fallback on earlier versions

}

return minZoomFactor

}

// 最大缩放值

func maxZoomFactor() -> CGFloat {

var maxZoomFactor = self.device?.activeFormat.videoMaxZoomFactor

if #available(iOS 11.0, \*) {

maxZoomFactor = self.device?.maxAvailableVideoZoomFactor

} else {

// Fallback on earlier versions

}

return maxZoomFactor!

}

private func saveImageToPhotoLibrary(image: UIImage?) {

guard let img = image else {

return

}

// 判断权限

switch PHPhotoLibrary.authorizationStatus() {

case .notDetermined: //用户尚未就此应用程序做出选择

PHPhotoLibrary.requestAuthorization { (status) in

if status == .authorized{

self.saveImage(image: img)

}else{

self.showAlert(title: "Please open album permissions", message: "Settings-Privacy-Photo")

}

}

case .restricted: //此应用程序无权访问照片数据。

self.showAlert(title: "Please open album permissions", message: "Settings-Privacy-Photo")

case .denied: //用户已明确拒绝此应用程序访问照片数据。

self.showAlert(title: "Please open album permissions", message: "Settings-Privacy-Photo")

case .authorized://用户已授权此应用程序访问照片数据。

saveImage(image: img)

case .limited://用户已授权此应用程序用于受限照片库访问。

break

default:

break

}

}

func saveImage(image: UIImage) -> Void {

PHPhotoLibrary.shared().performChanges {

PHAssetChangeRequest.creationRequestForAsset(from: image)

} completionHandler: { (success, error) in

DispatchQueue.main.async {

if success {

// 保存成功

SVProgressHUD.showSuccess(withStatus: NSLocalizedString("Saved to album", comment: ""))

}else {

SVProgressHUD.showError(withStatus: NSLocalizedString("Save failed", comment: ""))

}

}

}

}

func showAlert(title:String, message:String) -> Void {

// 延迟执行,解决 whose view is not in the window hierarchy! 问题

DispatchQueue.main.asyncAfter(deadline: .now() + 0) {

let alert = UIAlertController.init(title: title, message: message, preferredStyle: .alert)

let action = UIAlertAction.init(title: "OK", style: .default) { (action) in

}

alert.addAction(action)

self.present(alert, animated: true) {

}

}

}

/\*

// 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.

}

\*/

}

extension LBCameraViewController: SectionedSliderDelegate {

func sectionChanged(slider: SectionedSlider, selected: Int) {

print("light--> \(selected)")

if self.lightView.isHidden == false {

// 震动

let generator = UIImpactFeedbackGenerator(style: .medium)

generator.impactOccurred()

}else{

// Do noting

}

// 更改设置的时候必须先锁定设备，修改完后再解锁，否则崩溃

if ((device?.hasTorch) != nil) {

do {

try device?.lockForConfiguration()

if selected > 0 {

self.defaultLightButton.setImage(UIImage.init(named: "light\_selected"), for: .normal)

try! device?.setTorchModeOn(level: Float(selected)/10.0)

}else{

self.defaultLightButton.setImage(UIImage.init(named: "light\_default"), for: .normal)

device?.torchMode = AVCaptureDevice.TorchMode.off

}

device?.unlockForConfiguration()

} catch {

print(error)

}

}else{

// Do noting - 不支持手电筒

}

}

}

extension LBCameraViewController : AVCaptureVideoDataOutputSampleBufferDelegate {

func captureOutput(\_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection: AVCaptureConnection) {

if output == self.videoDataOutput { // 处理视频帧

// 处理图片

// print(sampleBuffer)

let imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)

var result = CIImage.init(cvPixelBuffer: imageBuffer!, options: nil)

// 添加滤镜

var filter : CIFilter?

switch self.currentFilterMode {

case .Yellow\_Black:

filter = CIFilter.init(name: "CIFalseColor")

filter?.setValue(CIColor.init(color: UIColor.yellow), forKey: "inputColor0")

filter?.setValue(CIColor.init(color: UIColor.black), forKey: "inputColor1")

case .Yellow\_Blue:

filter = CIFilter.init(name: "CIFalseColor")

filter?.setValue(CIColor.init(color: UIColor.yellow), forKey: "inputColor0")

filter?.setValue(CIColor.init(color: UIColor.blue), forKey: "inputColor1")

case .Reverse:

filter = CIFilter.init(name: "CIColorInvert")

case .Black\_White:

filter = CIFilter.init(name: "CIPhotoEffectNoir")

default:

filter = nil

}

if filter != nil {

filter?.setValue(result, forKey: kCIInputImageKey)

result = (filter?.outputImage)!

}else{

// Do noting

}

// 处理设备旋转镜像问题

let orientation = UIDevice.current.orientation

var transform : CGAffineTransform

if orientation == .portrait {

transform = CGAffineTransform(rotationAngle: -.pi / 2)

}else if orientation == .portraitUpsideDown {

transform = CGAffineTransform(rotationAngle: .pi / 2)

}else if orientation == .landscapeRight {

transform = CGAffineTransform(rotationAngle: .pi)

}else{

transform = CGAffineTransform(rotationAngle: 0.0)

}

result = result.transformed(by: transform)

let cgImage = CIContext.init().createCGImage(result, from: result.extent)

self.takeImage = UIImage.init(cgImage: cgImage!)

DispatchQueue.main.async {

self.previewLayer.contents = cgImage

}

}

}

}

import UIKit

class LBAboutTableViewController: UITableViewController {

override func viewDidLoad() {

super.viewDidLoad()

self.title = NSLocalizedString("About", comment: "")

// Uncomment the following line to preserve selection between presentations

// self.clearsSelectionOnViewWillAppear = false

// Uncomment the following line to display an Edit button in the navigation bar for this view controller.

// self.navigationItem.rightBarButtonItem = self.editButtonItem

}

// MARK: - Table view data source

override func tableView(\_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat {

return 10

}

override func tableView(\_ tableView: UITableView, heightForFooterInSection section: Int) -> CGFloat {

return 0.01

}

override func tableView(\_ tableView: UITableView, willDisplayHeaderView view: UIView, forSection section: Int) {

if view.isKind(of: UITableViewHeaderFooterView.self) {

view.tintColor = UIColor.clear

}

}

override func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

switch indexPath.section {

case 0:

switch indexPath.row {

case 0:

print("分享")

// 调用系统分享

// 分享内容

let shareTitle = NSLocalizedString("ZOOM 20X", comment: "")

let shareImage = UIImage.init(named: "logo")

let shareUrl = URL.init(string: "https://apps.apple.com/us/app/id\(AppStoreAPPID)")

let activityItemsArray : Array = [shareTitle, shareImage!, shareUrl!] as [Any]

// 调用分享

let activityVC = UIActivityViewController.init(activityItems: activityItemsArray, applicationActivities: nil)

activityVC.isModalInPopover = true

// iOS8.0 之后用此方法回调

activityVC.completionWithItemsHandler = { activityType, completed, items, error in

if completed {

print("分享成功")

}else{

print("分享失败")

}

}

self.present(activityVC, animated: true) {

}

default:

print("评分")

let urlStr = "itms-apps://itunes.apple.com/WebObjects/MZStore.woa/wa/viewContentsUserReviews?type=Purple+Software&id=\(AppStoreAPPID)&pageNumber=0&sortOrdering=2&mt=8"

UIApplication.shared.open(URL.init(string: urlStr)!) { (success) in

}

}

default:

switch indexPath.row {

case 0:

print("隐私")

let vc = LBWebViewController.init()

vc.url = WebUrl\_Privacy

vc.title = NSLocalizedString("Privacy Policy", comment: "")

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

case 1:

print("协议")

let vc = LBWebViewController.init()

vc.url = WebUrl\_Terms

vc.title = NSLocalizedString("Terms of Service", comment: "")

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

default:

break

}

}

}

}

import UIKit

typealias acceptButtonBlock = () -> Void

class LBPrivacyViewController: LBBaseViewController,UITextViewDelegate {

var acceptBlock : acceptButtonBlock?

@IBOutlet weak var textView: UITextView!

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.view.backgroundColor = UIColor.black.withAlphaComponent(0.5)

let str1 = NSLocalizedString("Privacy Detail1", comment: "")

let str2 = NSLocalizedString("Privacy Detail2", comment: "")

let str3 = NSLocalizedString("Privacy Detail3", comment: "")

let str4 = NSLocalizedString("Privacy Detail4", comment: "")

let str5 = NSLocalizedString("Privacy Detail5", comment: "")

let str:NSString = (str1 + str2 + str3 + str4 + str5) as NSString

let range1 = str.range(of: str2)

let range2 = str.range(of: str4)

let parStyle = NSMutableParagraphStyle()

parStyle.lineSpacing = 2

let att = NSMutableAttributedString(string: str as String, attributes: [NSAttributedString.Key.font:UIFont.systemFont(ofSize: 14),NSAttributedString.Key.foregroundColor:UIColor.hexString("#000000", a: 0.7),NSAttributedString.Key.paragraphStyle:parStyle])

let valueStr1 = "fwtk://\(str2)".addingPercentEncoding(withAllowedCharacters: CharacterSet.urlFragmentAllowed)

let valueStr2 = "yszc://\(str4)".addingPercentEncoding(withAllowedCharacters: CharacterSet.urlFragmentAllowed)

att.addAttributes([NSAttributedString.Key.link : valueStr1 as Any,], range: range1)

att.addAttributes([NSAttributedString.Key.link : valueStr2 as Any], range: range2)

textView.linkTextAttributes = [NSAttributedString.Key.foregroundColor:UIColor.hexString("#CB8C00", a: 1.0)]

textView.attributedText = att

}

// MARK: - UITextViewDelegate

func textView(\_ textView: UITextView, shouldInteractWith URL: URL, in characterRange: NSRange, interaction: UITextItemInteraction) -> Bool {

if URL.scheme == "fwtk" {

let vc = LBWebViewController.init()

vc.url = WebUrl\_Terms

vc.title = NSLocalizedString("Terms of Service", comment: "")

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

return false

}else if URL.scheme == "yszc"{

let vc = LBWebViewController.init()

vc.url = WebUrl\_Privacy

vc.title = NSLocalizedString("Privacy Policy", comment: "")

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

return false

}

return true

}

@IBAction func acceptButtonAction(\_ sender: UIButton) {

self.dismiss(animated: false) {

}

if self.acceptBlock != nil {

self.acceptBlock!()

}

}

/\*

// 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.

}

\*/

}

import UIKit

typealias BuySuccessBlock = () -> Void

class LBBuyVIPViewController: LBBaseViewController, UITextViewDelegate {

var BuyButtonSuccessBlock : BuySuccessBlock?

@IBOutlet weak var textView: UITextView!

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

let str1 = NSLocalizedString("Buy Detail1", comment: "")

let str2 = NSLocalizedString("Buy Detail2", comment: "")

let str3 = NSLocalizedString("Buy Detail3", comment: "")

let str4 = NSLocalizedString("Buy Detail4", comment: "")

let str5 = NSLocalizedString("Buy Detail5", comment: "")

let str:NSString = (str1 + str2 + str3 + str4 + str5) as NSString

let range1 = str.range(of: str2)

let range2 = str.range(of: str4)

let parStyle = NSMutableParagraphStyle()

parStyle.lineSpacing = 2

let att = NSMutableAttributedString(string: str as String, attributes: [NSAttributedString.Key.font:UIFont.systemFont(ofSize: 10),NSAttributedString.Key.foregroundColor:UIColor.hexString("#000000", a: 0.3),NSAttributedString.Key.paragraphStyle:parStyle])

let valueStr1 = "fwtk://\(str2)".addingPercentEncoding(withAllowedCharacters: CharacterSet.urlFragmentAllowed)

let valueStr2 = "yszc://\(str4)".addingPercentEncoding(withAllowedCharacters: CharacterSet.urlFragmentAllowed)

att.addAttributes([NSAttributedString.Key.link : valueStr1 as Any,], range: range1)

att.addAttributes([NSAttributedString.Key.link : valueStr2 as Any], range: range2)

textView.linkTextAttributes = [NSAttributedString.Key.foregroundColor:UIColor.hexString("#323232", a: 1.0)]

textView.attributedText = att

}

// MARK: - Action

@IBAction func closeButtonAction(\_ sender: UIButton) {

self.dismiss(animated: true) {

}

}

@IBAction func buyButtonAction(\_ sender: UIButton) {

// //通过product id 购买商品

// LBIAPTool.sharedInstance.payWithProduct(productID: "IAPDemo") { (finish) in

// if finish {

// // 购买成功

//

// if self.BuyButtonSuccessBlock != nil {

// self.BuyButtonSuccessBlock!()

// }

//

// self.dismiss(animated: true) {

//

// }

//

// }else{

// // 购买失败

// }

// }

// // 假数据，测试用

// UserDefaults.standard.set(1601282567000, forKey: "EXPIRE\_TIME")

// if self.BuyButtonSuccessBlock != nil {

// self.BuyButtonSuccessBlock!()

// }

//

// self.dismiss(animated: true) {

//

// }

}

@IBAction func restoreButtonAction(\_ sender: UIButton) {

// print("恢复购买")

// LBIAPTool.sharedInstance.restorePurchases { (finish) in

//

// }

// // 假数据，测试用

// UserDefaults.standard.set(1601282567000, forKey: "EXPIRE\_TIME")

// if self.BuyButtonSuccessBlock != nil {

// self.BuyButtonSuccessBlock!()

// }

//

// self.dismiss(animated: true) {

//

// }

}

// MARK: - UITextViewDelegate

func textView(\_ textView: UITextView, shouldInteractWith URL: URL, in characterRange: NSRange, interaction: UITextItemInteraction) -> Bool {

if URL.scheme == "fwtk" {

let vc = LBWebViewController.init()

vc.url = WebUrl\_Terms

vc.title = NSLocalizedString("Terms of Service", comment: "")

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

return false

}else if URL.scheme == "yszc"{

let vc = LBWebViewController.init()

vc.url = WebUrl\_Privacy

vc.title = NSLocalizedString("Privacy Policy", comment: "")

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

return false

}

return true

}

/\*

// 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.

}

\*/

}

import UIKit

class LBBaseViewController: UIViewController {

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

}

override var preferredStatusBarStyle: UIStatusBarStyle{

// 白色

return .lightContent

// // 黑色

// if #available(iOS 13.0, \*) {

// return .darkContent

// } else {

// // Fallback on earlier versions

// return .default

// }

}

/\*

// 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.

}

\*/

}

import UIKit

class LBBaseNavigationController: UINavigationController {

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

navigationBar.setBackgroundImage(UIImage.Create(size: CGSize(width: 1, height: 1), color: .white), for: .default)

navigationBar.shadowImage = UIImage()

let line = UIView.init(frame: CGRect(x: 0, y: self.navigationBar.height - 0.33, width: SCREEN\_WIDTH, height: 0.33))

line.backgroundColor = UIColor.hexString("#000000", a: 0.1)

navigationBar.addSubview(line)

}

override var preferredStatusBarStyle: UIStatusBarStyle{

let topVC = self.topViewController

return topVC!.preferredStatusBarStyle

}

/\*

// 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.

}

\*/

}

import UIKit

public protocol SectionedSliderDelegate {

func sectionChanged(slider: SectionedSlider, selected: Int)

}

public class StyleKit : NSObject {

//// Drawing Methods

public static func drawSlider(frame targetFrame: CGRect = CGRect(x: 0, y: 0, width: 156, height: 400), resizing: ResizingBehavior = .stretch, factor: CGFloat = 0.0, sections: CGFloat = 10, palette: Palette = Palette()) {

//// General Declarations

let context = UIGraphicsGetCurrentContext()!

let sectionOriginalHeight: CGFloat = 400

let sectionOriginalWidth: CGFloat = 156

//// Resize to Target Frame

context.saveGState()

let resizedFrame: CGRect = resizing.apply(rect: CGRect(x: 0, y: 0, width: sectionOriginalWidth, height: sectionOriginalHeight), target: targetFrame)

context.translateBy(x: resizedFrame.minX, y: resizedFrame.minY)

context.scaleBy(x: resizedFrame.width / sectionOriginalWidth, y: resizedFrame.height / sectionOriginalHeight)

//// Variable Declarations

let sectionsSafe: CGFloat = sections < 2 ? 2 : (sections > 20 ? 20 : sections)

let sectionHeight: CGFloat = sectionOriginalHeight / sectionsSafe

let slideHeight: CGFloat = floor(factor / (1.0 / sectionsSafe) + 1) \* sectionHeight

let y1: CGFloat = sectionHeight \* 0 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 0

let y2: CGFloat = sectionHeight \* 1 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 1

let y3: CGFloat = sectionHeight \* 2 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 2

let y4: CGFloat = sectionHeight \* 3 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 3

let y5: CGFloat = sectionHeight \* 4 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 4

let y6: CGFloat = sectionHeight \* 5 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 5

let y7: CGFloat = sectionHeight \* 6 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 6

let y8: CGFloat = sectionHeight \* 7 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 7

let y9: CGFloat = sectionHeight \* 8 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 8

let y10: CGFloat = sectionHeight \* 9 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 9

let y11: CGFloat = sectionHeight \* 10 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 10

let y12: CGFloat = sectionHeight \* 11 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 11

let y13: CGFloat = sectionHeight \* 12 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 12

let y14: CGFloat = sectionHeight \* 13 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 13

let y15: CGFloat = sectionHeight \* 14 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 14

let y16: CGFloat = sectionHeight \* 15 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 15

let y17: CGFloat = sectionHeight \* 16 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 16

let y18: CGFloat = sectionHeight \* 17 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 17

let y19: CGFloat = sectionHeight \* 18 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 18

let y20: CGFloat = sectionHeight \* 19 > sectionOriginalHeight + 0.00000001 - sectionHeight ? 0 : sectionHeight \* 19

//// BackgroundView Drawing

let backgroundViewPath = UIBezierPath(rect: CGRect(x: 0, y: 0, width: sectionOriginalWidth, height: sectionOriginalHeight))

palette.viewBackgroundColor.setFill()

backgroundViewPath.fill()

//// BodyBackgroundView Drawing

let bodyBackgroundViewPath = UIBezierPath()

bodyBackgroundViewPath.move(to: CGPoint(x: 76.43, y: 0))

bodyBackgroundViewPath.addLine(to: CGPoint(x: 79.57, y: 0))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 122.5, y: 3.27), controlPoint1: CGPoint(x: 101.58, y: 0), controlPoint2: CGPoint(x: 112.58, y: 0))

bodyBackgroundViewPath.addLine(to: CGPoint(x: 124.43, y: 3.75))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 152.25, y: 31.57), controlPoint1: CGPoint(x: 137.36, y: 8.45), controlPoint2: CGPoint(x: 147.55, y: 18.64))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 156, y: 76.43), controlPoint1: CGPoint(x: 156, y: 43.42), controlPoint2: CGPoint(x: 156, y: 54.42))

bodyBackgroundViewPath.addLine(to: CGPoint(x: 156, y: 323.57))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 152.73, y: 366.5), controlPoint1: CGPoint(x: 156, y: 345.58), controlPoint2: CGPoint(x: 156, y: 356.58))

bodyBackgroundViewPath.addLine(to: CGPoint(x: 152.25, y: 368.43))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 124.43, y: 396.25), controlPoint1: CGPoint(x: 147.55, y: 381.36), controlPoint2: CGPoint(x: 137.36, y: 391.55))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 79.57, y: 400), controlPoint1: CGPoint(x: 112.58, y: 400), controlPoint2: CGPoint(x: 101.58, y: 400))

bodyBackgroundViewPath.addLine(to: CGPoint(x: 76.43, y: 400))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 33.5, y: 396.73), controlPoint1: CGPoint(x: 54.42, y: 400), controlPoint2: CGPoint(x: 43.42, y: 400))

bodyBackgroundViewPath.addLine(to: CGPoint(x: 31.57, y: 396.25))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 3.75, y: 368.43), controlPoint1: CGPoint(x: 18.64, y: 391.55), controlPoint2: CGPoint(x: 8.45, y: 381.36))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 0, y: 323.57), controlPoint1: CGPoint(x: 0, y: 356.58), controlPoint2: CGPoint(x: 0, y: 345.58))

bodyBackgroundViewPath.addLine(to: CGPoint(x: 0, y: 76.43))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 3.27, y: 33.5), controlPoint1: CGPoint(x: 0, y: 54.42), controlPoint2: CGPoint(x: 0, y: 43.42))

bodyBackgroundViewPath.addLine(to: CGPoint(x: 3.75, y: 31.57))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 31.57, y: 3.75), controlPoint1: CGPoint(x: 8.45, y: 18.64), controlPoint2: CGPoint(x: 18.64, y: 8.45))

bodyBackgroundViewPath.addCurve(to: CGPoint(x: 76.43, y: 0), controlPoint1: CGPoint(x: 43.42, y: -0), controlPoint2: CGPoint(x: 54.42, y: 0))

bodyBackgroundViewPath.close()

palette.sliderBackgroundColor.setFill()

bodyBackgroundViewPath.fill()

//// Group

context.saveGState()

context.beginTransparencyLayer(auxiliaryInfo: nil)

//// SectionedMask Drawing

context.saveGState()

context.translateBy(x: 78, y: 200)

context.rotate(by: -180 \* CGFloat.pi/180)

let sectionedMaskPath = UIBezierPath(rect: CGRect(x: -78, y: -200, width: sectionOriginalWidth, height: slideHeight))

palette.sliderColor.setFill()

sectionedMaskPath.fill()

context.restoreGState()

//// BodyView Drawing

context.saveGState()

context.setBlendMode(.sourceIn)

context.beginTransparencyLayer(auxiliaryInfo: nil)

// let bodyViewPath = UIBezierPath()

// bodyViewPath.move(to: CGPoint(x: 76.43, y: 0))

// bodyViewPath.addLine(to: CGPoint(x: 79.57, y: 0))

// bodyViewPath.addCurve(to: CGPoint(x: 122.5, y: 3.27), controlPoint1: CGPoint(x: 101.58, y: 0), controlPoint2: CGPoint(x: 112.58, y: -0))

// bodyViewPath.addLine(to: CGPoint(x: 124.43, y: 3.75))

// bodyViewPath.addCurve(to: CGPoint(x: 152.25, y: 31.57), controlPoint1: CGPoint(x: 137.36, y: 8.45), controlPoint2: CGPoint(x: 147.55, y: 18.64))

// bodyViewPath.addCurve(to: CGPoint(x: 156, y: 76.43), controlPoint1: CGPoint(x: 156, y: 43.42), controlPoint2: CGPoint(x: 156, y: 54.42))

// bodyViewPath.addLine(to: CGPoint(x: 156, y: 323.57))

// bodyViewPath.addCurve(to: CGPoint(x: 152.73, y: 366.5), controlPoint1: CGPoint(x: 156, y: 345.58), controlPoint2: CGPoint(x: 156, y: 356.58))

// bodyViewPath.addLine(to: CGPoint(x: 152.25, y: 368.43))

// bodyViewPath.addCurve(to: CGPoint(x: 124.43, y: 396.25), controlPoint1: CGPoint(x: 147.55, y: 381.36), controlPoint2: CGPoint(x: 137.36, y: 391.55))

// bodyViewPath.addCurve(to: CGPoint(x: 79.57, y: 400), controlPoint1: CGPoint(x: 112.58, y: 400), controlPoint2: CGPoint(x: 101.58, y: 400))

// bodyViewPath.addLine(to: CGPoint(x: 76.43, y: 400))

// bodyViewPath.addCurve(to: CGPoint(x: 33.5, y: 396.73), controlPoint1: CGPoint(x: 54.42, y: 400), controlPoint2: CGPoint(x: 43.42, y: 400))

// bodyViewPath.addLine(to: CGPoint(x: 31.57, y: 396.25))

// bodyViewPath.addCurve(to: CGPoint(x: 3.75, y: 368.43), controlPoint1: CGPoint(x: 18.64, y: 391.55), controlPoint2: CGPoint(x: 8.45, y: 381.36))

// bodyViewPath.addCurve(to: CGPoint(x: 0, y: 323.57), controlPoint1: CGPoint(x: 0, y: 356.58), controlPoint2: CGPoint(x: 0, y: 345.58))

// bodyViewPath.addLine(to: CGPoint(x: 0, y: 76.43))

// bodyViewPath.addCurve(to: CGPoint(x: 3.27, y: 33.5), controlPoint1: CGPoint(x: 0, y: 54.42), controlPoint2: CGPoint(x: 0, y: 43.42))

// bodyViewPath.addLine(to: CGPoint(x: 3.75, y: 31.57))

// bodyViewPath.addCurve(to: CGPoint(x: 31.57, y: 3.75), controlPoint1: CGPoint(x: 8.45, y: 18.64), controlPoint2: CGPoint(x: 18.64, y: 8.45))

// bodyViewPath.addCurve(to: CGPoint(x: 76.43, y: 0), controlPoint1: CGPoint(x: 43.42, y: -0), controlPoint2: CGPoint(x: 54.42, y: 0))

// bodyViewPath.close()

palette.sliderColor.setFill()

// bodyViewPath.fill()

context.endTransparencyLayer()

context.restoreGState()

//// Rectangle 20 Drawing

let rectangle20Path = UIBezierPath(rect: CGRect(x: 0, y: y20, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle20Path.lineWidth = 1

rectangle20Path.stroke()

//// Rectangle 19 Drawing

let rectangle19Path = UIBezierPath(rect: CGRect(x: 0, y: y19, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle19Path.lineWidth = 1

rectangle19Path.stroke()

//// Rectangle 18 Drawing

let rectangle18Path = UIBezierPath(rect: CGRect(x: 0, y: y18, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle18Path.lineWidth = 1

rectangle18Path.stroke()

//// Rectangle 17 Drawing

let rectangle17Path = UIBezierPath(rect: CGRect(x: 0, y: y17, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle17Path.lineWidth = 1

rectangle17Path.stroke()

//// Rectangle 16 Drawing

let rectangle16Path = UIBezierPath(rect: CGRect(x: 0, y: y16, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle16Path.lineWidth = 1

rectangle16Path.stroke()

//// Rectangle 15 Drawing

let rectangle15Path = UIBezierPath(rect: CGRect(x: 0, y: y15, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle15Path.lineWidth = 1

rectangle15Path.stroke()

//// Rectangle 14 Drawing

let rectangle14Path = UIBezierPath(rect: CGRect(x: 0, y: y14, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle14Path.lineWidth = 1

rectangle14Path.stroke()

//// Rectangle 13 Drawing

let rectangle13Path = UIBezierPath(rect: CGRect(x: 0, y: y13, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle13Path.lineWidth = 1

rectangle13Path.stroke()

//// Rectangle 12 Drawing

let rectangle12Path = UIBezierPath(rect: CGRect(x: 0, y: y12, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle12Path.lineWidth = 1

rectangle12Path.stroke()

//// Rectangle 11 Drawing

let rectangle11Path = UIBezierPath(rect: CGRect(x: 0, y: y11, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle11Path.lineWidth = 1

rectangle11Path.stroke()

//// Rectangle 10 Drawing

let rectangle10Path = UIBezierPath(rect: CGRect(x: 0, y: y10, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle10Path.lineWidth = 1

rectangle10Path.stroke()

//// Rectangle 9 Drawing

let rectangle9Path = UIBezierPath(rect: CGRect(x: 0, y: y9, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle9Path.lineWidth = 1

rectangle9Path.stroke()

//// Rectangle 8 Drawing

let rectangle8Path = UIBezierPath(rect: CGRect(x: 0, y: y8, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle8Path.lineWidth = 1

rectangle8Path.stroke()

//// Rectangle 7 Drawing

let rectangle7Path = UIBezierPath(rect: CGRect(x: 0, y: y7, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle7Path.lineWidth = 1

rectangle7Path.stroke()

//// Rectangle 6 Drawing

let rectangle6Path = UIBezierPath(rect: CGRect(x: 0, y: y6, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle6Path.lineWidth = 1

rectangle6Path.stroke()

//// Rectangle 5 Drawing

let rectangle5Path = UIBezierPath(rect: CGRect(x: 0, y: y5, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle5Path.lineWidth = 1

rectangle5Path.stroke()

//// Rectangle 4 Drawing

let rectangle4Path = UIBezierPath(rect: CGRect(x: 0, y: y4, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle4Path.lineWidth = 1

rectangle4Path.stroke()

//// Rectangle 3 Drawing

let rectangle3Path = UIBezierPath(rect: CGRect(x: 0, y: y3, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle3Path.lineWidth = 1

rectangle3Path.stroke()

//// Rectangle 2 Drawing

let rectangle2Path = UIBezierPath(rect: CGRect(x: 0, y: y2, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle2Path.lineWidth = 1

rectangle2Path.stroke()

//// Rectangle 1 Drawing

let rectangle1Path = UIBezierPath(rect: CGRect(x: 0, y: y1, width: sectionOriginalWidth, height: sectionHeight))

palette.viewBackgroundColor.setStroke()

rectangle1Path.lineWidth = 1

rectangle1Path.stroke()

context.saveGState()

context.setBlendMode(.multiply)

context.restoreGState()

context.endTransparencyLayer()

context.restoreGState()

context.restoreGState()

}

@objc(SectionedSliderResizingBehavior)

public enum ResizingBehavior: Int {

case aspectFit /// The content is proportionally resized to fit into the target rectangle.

case aspectFill /// The content is proportionally resized to completely fill the target rectangle.

case stretch /// The content is stretched to match the entire target rectangle.

case center /// The content is centered in the target rectangle, but it is NOT resized.

public func apply(rect: CGRect, target: CGRect) -> CGRect {

if rect == target || target == CGRect.zero {

return rect

}

var scales = CGSize.zero

scales.width = abs(target.width / rect.width)

scales.height = abs(target.height / rect.height)

switch self {

case .aspectFit:

scales.width = min(scales.width, scales.height)

scales.height = scales.width

case .aspectFill:

scales.width = max(scales.width, scales.height)

scales.height = scales.width

case .stretch:

break

case .center:

scales.width = 1

scales.height = 1

}

var result = rect.standardized

result.size.width \*= scales.width

result.size.height \*= scales.height

result.origin.x = target.minX + (target.width - result.width) / 2

result.origin.y = target.minY + (target.height - result.height) / 2

return result

}

}

}

class SectionedSliderLayer: CALayer {

// MARK: - Properties

@NSManaged var factor: CGFloat

// MARK: - Initializers

override init() {

super.init()

factor = 0

}

override init(layer: Any) {

super.init(layer: layer)

if let layer = layer as? SectionedSliderLayer {

factor = layer.factor

}

}

required init?(coder aDecoder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

}

internal class Flow {

// MARK: - Functions

// Execute code block asynchronously

static func async(block: @escaping () -> Void) {

DispatchQueue.main.async(execute: block)

}

// Execute code block asynchronously after given delay time

static func delay(for delay: TimeInterval, block: @escaping () -> Void) {

DispatchQueue.main.asyncAfter(deadline: DispatchTime.now() + delay, execute: block)

}

}

open class Palette {

var viewBackgroundColor: UIColor = UIColor(red: 0.000, green: 0.000, blue: 0.000, alpha: 1.000)

var sliderBackgroundColor: UIColor = UIColor(red: 0.185, green: 0.184, blue: 0.184, alpha: 1.000)

var sliderColor: UIColor = UIColor(red: 0.147, green: 0.000, blue: 0.697, alpha: 1.000)

public init(viewBackgroundColor: UIColor? = nil, sliderBackgroundColor: UIColor? = nil, sliderColor: UIColor? = nil) {

self.viewBackgroundColor = viewBackgroundColor ?? self.viewBackgroundColor

self.sliderBackgroundColor = sliderBackgroundColor ?? self.sliderBackgroundColor

self.sliderColor = sliderColor ?? self.sliderColor

}

}

//@IBDesignable

open class SectionedSlider: UIView {

// MARK: - IBDesignable and IBInspectable

@IBInspectable var viewBackgroundColor: UIColor? {

didSet {

palette.viewBackgroundColor = viewBackgroundColor ?? palette.viewBackgroundColor

}

}

@IBInspectable var sliderBackgroundColor: UIColor? {

didSet {

palette.sliderBackgroundColor = sliderBackgroundColor ?? palette.sliderBackgroundColor

}

}

@IBInspectable var sliderColor: UIColor? {

didSet {

palette.sliderColor = sliderColor ?? palette.sliderColor

}

}

@IBInspectable open var sections: Int = 10 {

willSet {

if newValue < 2 || newValue > 20 {

debugPrint("sections must be between 2 and 20")

}

}

}

@IBInspectable open var selectedSection: Int = 0 {

didSet {

if selectedSection < 0 || selectedSection > sections {

debugPrint("sections must be between 0 and \(sections)")

} else {

factor = CGFloat(selectedSection) / CGFloat(sections) - 0.0001

}

}

}

private var factor: CGFloat = 0.0 {

willSet {

(layer as? SectionedSliderLayer)?.factor = newValue

delegate?.sectionChanged(slider: self, selected: abs(Int(ceil(CGFloat(newValue) \* CGFloat(sections)))))

draw()

}

}

// MARK: - Properties

private var keyPath: String = "factor"

private var palette: Palette = Palette()

open var delegate: SectionedSliderDelegate? {

didSet {

let factor = self.factor

self.factor = factor

}

}

override open class var layerClass: AnyClass {

return SectionedSliderLayer.self

}

override init(frame: CGRect) {

super.init(frame: frame)

}

public init(frame: CGRect, selectedSection: Int, sections: Int, palette: Palette = Palette()) {

super.init(frame: frame)

//Because we use observers, for them to run in the initializers defer is needed.

// See: https://stackoverflow.com/a/33979852/1904232

defer {

self.backgroundColor = palette.viewBackgroundColor

self.sections = sections

self.selectedSection = selectedSection

self.palette = palette

}

addPanGesture()

draw()

}

required public init?(coder aDecoder: NSCoder) {

super.init(coder: aDecoder)

}

// MARK: - Lifecyle

override open func awakeFromNib() {

super.awakeFromNib()

addPanGesture()

draw()

}

override open func draw(\_ layer: CALayer, in ctx: CGContext) {

guard let layer: SectionedSliderLayer = layer as? SectionedSliderLayer else { return }

let frame = CGRect(origin: CGPoint(x: 0, y: 0), size: layer.frame.size)

UIGraphicsPushContext(ctx)

switch keyPath {

case "factor":

StyleKit.drawSlider(frame: frame, factor: layer.factor, sections: CGFloat(sections), palette: palette)

break

default:

break

}

UIGraphicsPopContext()

}

// MARK: - Functions

private func addPanGesture() {

let gesture = UIPanGestureRecognizer(target: self, action: #selector(SectionedSlider.dragged(gesture:)))

self.addGestureRecognizer(gesture)

}

private func resetManipulables() {

guard let layer: SectionedSliderLayer = layer as? SectionedSliderLayer else { return }

layer.factor = 0.0

}

func draw() {

needsDisplay()

}

func needsDisplay() {

layer.contentsScale = UIScreen.main.scale

layer.setNeedsDisplay()

}

open override func touchesBegan(\_ touches: Set<UITouch>, with event: UIEvent?) {

super.touchesBegan(touches, with: event)

guard

let touch = touches.first

else { return }

var x = self.frame.height - touch.location(in: self).y

x = x < 0 ? -1 : (x > self.frame.height ? self.frame.height : x)

factor = x / self.frame.height

}

@objc private func dragged(gesture: UIPanGestureRecognizer) {

let point = gesture.location(in: self)

var x = self.frame.height - point.y

x = x < 0 ? -1 : (x > self.frame.height ? self.frame.height : x)

factor = x / self.frame.height

}

}