**import UIKit**

**import SwiftyJSON**

**class CoutryModel:NSObject{**

**var FIFA = ""**

**var Dial = ""**

**var ISO3166\_1\_Alpha\_3 = ""**

**var MARC = ""**

**var is\_independent\_ = ""**

**var ISO3166\_1\_numeric = ""**

**var GAUL = ""**

**var FIPS = ""**

**var WMO = ""**

**var ISO3166\_1\_Alpha\_2 = ""**

**var ITU = ""**

**var IOC = ""**

**var DS = ""**

**var UNTERM\_Spanish\_Formal = ""**

**var Global\_Code = ""**

**var Intermediate\_Region\_Code = ""**

**var official\_name\_fr = ""**

**var UNTERM\_French\_Short = ""**

**var ISO4217\_currency\_name = ""**

**var Developed\_Developing\_Countries = ""**

**var UNTERM\_Russian\_Formal = ""**

**var UNTERM\_English\_Short = ""**

**var ISO4217\_currency\_alphabetic\_code = ""**

**var Small\_Island\_Developing\_States\_SIDS = ""**

**var UNTERM\_Spanish\_Short = ""**

**var ISO4217\_currency\_numeric\_code = ""**

**var UNTERM\_Chinese\_Formal = ""**

**var UNTERM\_French\_Formal = ""**

**var UNTERM\_Russian\_Short = ""**

**var M49 = ""**

**var Sub\_region\_Code = ""**

**var Region\_Code = ""**

**var official\_name\_ar = ""**

**var ISO4217\_currency\_minor\_unit = ""**

**var UNTERM\_Arabic\_Formal = ""**

**var UNTERM\_Chinese\_Short = ""**

**var Land\_Locked\_Developing\_Countries\_LLDC = ""**

**var Intermediate\_Region\_Name = ""**

**var official\_name\_es = ""**

**var UNTERM\_English\_Formal = ""**

**var official\_name\_cn = ""**

**var official\_name\_en = ""**

**var ISO4217\_currency\_country\_name = ""**

**var Least\_Developed\_Countries\_LDC = ""**

**var Region\_Name = ""**

**var UNTERM\_Arabic\_Short = ""**

**var Sub\_region\_Name = ""**

**var official\_name\_ru = ""**

**var Global\_Name = ""**

**var Capital = ""**

**var Continent\_ = ""**

**var TLD = ""**

**var Languages = ""**

**var Geoname\_ID = ""**

**var CLDR\_display\_name = ""**

**var EDGAR = ""**

**init(json:JSON) {**

**self.FIFA = json["FIFA"].stringValue**

**self.Dial = json["Dial"].stringValue**

**self.ISO3166\_1\_Alpha\_3 = json["ISO3166-1-Alpha-3"].stringValue**

**self.MARC = json["MARC"].stringValue**

**self.is\_independent\_ = json["is\_independent"].stringValue**

**self.ISO3166\_1\_numeric = json["ISO3166-1-numeric"].stringValue**

**self.GAUL = json["GAUL"].stringValue**

**self.FIPS = json["FIPS"].stringValue**

**self.WMO = json["WMO"].stringValue**

**self.ISO3166\_1\_Alpha\_2 = json["ISO3166-1-Alpha-2"].stringValue**

**self.ITU = json["ITU"].stringValue**

**self.IOC = json["IOC"].stringValue**

**self.DS = json["DS"].stringValue**

**self.UNTERM\_Spanish\_Formal = json["UNTERM Spanish Formal"].stringValue**

**self.Global\_Code = json["Global Code"].stringValue**

**self.Intermediate\_Region\_Code = json["Intermediate Region Code"].stringValue**

**self.official\_name\_fr = json["official\_name\_fr"].stringValue**

**self.UNTERM\_French\_Short = json["UNTERM French Short"].stringValue**

**self.ISO4217\_currency\_name = json["ISO4217-currency\_name"].stringValue**

**self.Developed\_Developing\_Countries = json["Developed / Developing Countries"].stringValue**

**self.UNTERM\_Russian\_Formal = json["UNTERM Russian Formal"].stringValue**

**self.UNTERM\_English\_Short = json["UNTERM English Short"].stringValue**

**self.ISO4217\_currency\_alphabetic\_code = json["ISO4217-currency\_alphabetic\_code"].stringValue**

**self.Small\_Island\_Developing\_States\_SIDS = json["Small Island Developing States (SIDS)"].stringValue**

**self.UNTERM\_Spanish\_Short = json["UNTERM Spanish Short"].stringValue**

**self.ISO4217\_currency\_numeric\_code = json["ISO4217-currency\_numeric\_code"].stringValue**

**self.UNTERM\_Chinese\_Formal = json["UNTERM Chinese Formal"].stringValue**

**self.UNTERM\_French\_Formal = json["UNTERM French Formal"].stringValue**

**self.UNTERM\_Russian\_Short = json["UNTERM Russian Short"].stringValue**

**self.M49 = json["M49"].stringValue**

**self.Sub\_region\_Code = json["Sub-region Code"].stringValue**

**self.Region\_Code = json["Region Code"].stringValue**

**self.official\_name\_ar = json["official\_name\_ar"].stringValue**

**self.ISO4217\_currency\_minor\_unit = json["ISO4217-currency\_minor\_unit"].stringValue**

**self.UNTERM\_Arabic\_Formal = json["UNTERM Arabic Formal"].stringValue**

**self.UNTERM\_Chinese\_Short = json["UNTERM Chinese Short"].stringValue**

**self.Land\_Locked\_Developing\_Countries\_LLDC = json["Land Locked Developing Countries (LLDC)"].stringValue**

**self.Intermediate\_Region\_Name = json["Intermediate Region Name"].stringValue**

**self.official\_name\_es = json["official\_name\_es"].stringValue**

**self.UNTERM\_English\_Formal = json["UNTERM English Formal"].stringValue**

**self.official\_name\_cn = json["official\_name\_cn"].stringValue**

**self.official\_name\_en = json["official\_name\_en"].stringValue**

**self.ISO4217\_currency\_country\_name = json["ISO4217-currency\_country\_name"].stringValue**

**self.Least\_Developed\_Countries\_LDC = json["Least Developed Countries (LDC)"].stringValue**

**self.Region\_Name = json["Region Name"].stringValue**

**self.UNTERM\_Arabic\_Short = json["UNTERM Arabic Short"].stringValue**

**self.Sub\_region\_Name = json["Sub-region Name"].stringValue**

**self.official\_name\_ru = json["official\_name\_ru"].stringValue**

**self.Global\_Name = json["Global Name"].stringValue**

**self.Capital = json["Capital"].stringValue**

**self.Continent\_ = json["Continent"].stringValue**

**self.TLD = json["TLD"].stringValue**

**self.Languages = json["Languages"].stringValue**

**self.Geoname\_ID = json["Geoname ID"].stringValue**

**self.CLDR\_display\_name = json["CLDR display name"].stringValue**

**self.EDGAR = json["EDGAR"].stringValue**

**}**

**}**

**class MN\_CountryManager: NSObject {**

**class func MN\_getCountryModels() -> Array<CoutryModel>{**

**var resArr:Array<CoutryModel> = []**

**let path = Bundle.main.path(forResource: "country", ofType: "json")**

**if let jsonPath = path {**

**let data = NSData(contentsOfFile: jsonPath)**

**guard let countryArray = try? JSON(data: data! as Data) else {**

**return resArr**

**}**

**for json in countryArray.arrayValue {**

**let model = CoutryModel(json: json)**

**resArr.append(model)**

**}**

**}**

**return resArr**

**}**

**class func MN\_getCountryCategoryList() -> Dictionary<String,Array<CoutryModel>>{**

**var dic:Dictionary<String,Array<CoutryModel>> = [:]**

**let modelArr = MN\_getCountryModels()**

**for model in modelArr {**

**let str = "\(model.CLDR\_display\_name.first ?? " ")"**

**if dic.keys.contains("\(str)") {**

**dic[str]?.append(model)**

**}else{**

**dic[str] = [model]**

**}**

**dic[str]?.sorted(by: {$0.CLDR\_display\_name < $1.CLDR\_display\_name})**

**}**

**dic.sorted(by: {$0.key < $1.key})**

**return dic**

**}**

**}**

**import UIKit**

**import CallKit**

**enum CallStatus:Int {**

**case call = 8**

**case refuse = 9**

**case link = 10**

**case hungUp = 11**

**case newCall = 0**

**case holdAndAnswer = 14**

**case anotherHangsUp = 2**

**case otherHangsUp = 3**

**}**

**class MN\_CallStatusTool: AppDelegate ,CXCallObserverDelegate{**

**static let share = MN\_CallStatusTool()**

**var callObserver:CXCallObserver?**

**var status:CallStatus?**

**func applicationDidFinishLaunching(\_ application: UIApplication) {**

**callObserver = CXCallObserver()**

**callObserver?.setDelegate(self, queue: DispatchQueue.main)**

**}**

**func callObserver(\_ callObserver: CXCallObserver, callChanged call: CXCall) {**

**let str = call.isOutgoing.toString() + call.isOnHold.toString() + call.hasConnected.toString() + call.hasEnded.toString()**

**status = CallStatus(rawValue: str.getDecimalByBinary())**

**switch status {**

**case .call:**

**break**

**case .refuse:**

**break**

**case .link:**

**if currentCallType == . centerNumber && newCall == false{**

**MN\_UNNotificationTool.share.createNotificationContent()**

**}**

**break**

**case .hungUp:**

**currentCallType = . centerNumber**

**userCallNum = ""**

**break**

**case .newCall:**

**newCall = true**

**let vc = MN\_InCallRecordView.loadFromNib()**

**vc.frame = CGRect(x: 0, y: 0, width: LJScreenWidth, height: LJScreenHeight)**

**UIApplication.shared.keyWindow?.rootViewController?.view.addSubview(vc)**

**break**

**case .holdAndAnswer:**

**currentCallType = . userNumber**

**break**

**case .anotherHangsUp:**

**currentCallType = . centerNumber**

**break**

**case .otherHangsUp:**

**currentCallType = . centerNumber**

**break**

**default:**

**break**

**}**

**}**

**func applicationWillTerminate(\_ application: UIApplication) {**

**callObserver?.setDelegate(nil, queue: DispatchQueue.main)**

**callObserver = nil**

**}**

**}**

**import UIKit**

**import Alamofire**

**import SwiftyJSON**

**let appid = "3"**

**let appSecret = "jerwiefkjfd112AAdsdTRGssdWEsdf"**

**let BaseUrl = "https://xj.52yqs.com/iosapps"**

**let UserExit = "/v3/voice/record/user/exist/"**

**let SendCode = "/v3/verify/code/send/"**

**let VerifyCode = "/v3/verify/code/verify/"**

**let RecordList = "/v3/voice/record/list/"**

**let ChangeName = "/v3/voice/record/change\_name/"**

**let UserExpire = "/v3/voice/user/expire/"**

**let CenterNum = "/v3/voice/number/"**

**let UpdateUser = "/v3/voice/user/update/"**

**let DownloadRecording = "/v3/voice/record/download/"**

**let DeleteRecords = "/v3/voice/records/remove/"**

**let UploadEent = "/v3/data/"**

**class MN\_NetworkingTool: NSObject {**

**class func verifyReceipt(receiptStr:String,forceupdate:Bool?=false,success:((Response)->())? = nil,failure:((Error)->())? = nil){**

**let ts = Date().milliStamp**

**let nonce = String.randomStr()**

**let postBodyJson:JSON =  ["receipt":receiptStr]**

**let uuid = FCUUID.uuidForDevice() ?? ""**

**let bodyStr = self.jsonToString(json: postBodyJson)**

**let sig = (ts + nonce + bodyStr + appSecret).sha1()**

**let requestUrl = BaseUrl + "/v3/verify/receipt/" + appid + "/" + ts + "/" + nonce + "/" + sig + "/" + uuid + "/?force\_update=\(String(describing: forceupdate ?? false))"**

**let url = URL(string: requestUrl)!**

**var request = URLRequest(url: url)**

**request.httpMethod = HTTPMethod.post.rawValue**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**let requsetBody = try? JSONSerialization.data(withJSONObject: postBodyJson.dictionaryObject ?? [:], options: [])**

**request.httpBody = requsetBody**

**AF.request(request).responseString { response in**

**switch response.result{**

**case .success(\_):**

**if response.response?.statusCode == 200 ,let value = response.value{**

**UserDefaults.standard.setValue(receiptStr, forKey: "receiptStr")**

**let json = JSON.init(parseJSON: value )**

**let result = Response(json: json)**

**result.code = response.response?.statusCode ?? 0**

**success?(result)**

**}else{**

**let result = Response(json: JSON.init(parseJSON: response.value ?? ""))**

**result.code = response.response?.statusCode ?? 0**

**success?(result)**

**}**

**break**

**case .failure(let error):**

**failure?(error)**

**break;**

**}**

**}**

**}**

**class func requestWithGET(requestUrl:String,param:Dictionary<String,Any>? = nil,success:((Response) -> ())? = nil,failure:((Error)->())? = nil)  {**

**let ts = Date().milliStamp**

**let nonce = String.randomStr()**

**let sig = (ts + nonce + appSecret).sha1()**

**let newparam = param?.sorted(by: { Int($0.key) ?? 0 < Int($1.key) ?? 0**

**})**

**let parStr = newparam?.map({ return "\($0.value)"}).joined(separator: "/") ?? ""**

**let requestUrl = (BaseUrl + requestUrl  + parStr + "/" + appid + "/" + ts + "/" + nonce + "/"  + sig  + "/").addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)**

**let url = URL(string: requestUrl!)!**

**var request = URLRequest(url: url)**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**request.httpMethod = "GET"**

**AF.request(request).responseString { response in**

**switch response.result{**

**case .success(\_):**

**if response.response?.statusCode == 200 ,let value = response.value{**

**let json = JSON.init(parseJSON: value )**

**let result = Response(json: json)**

**result.code = response.response?.statusCode ?? 0**

**success?(result)**

**}else{**

**let result = Response(json: JSON.init(parseJSON: response.value ?? ""))**

**result.code = response.response?.statusCode ?? 0**

**success?(result)**

**}**

**break**

**case .failure(let error):**

**failure?(error)**

**break;**

**}**

**}**

**}**

**class func requestWithPOST(requestUrl:String,param:Dictionary<String,Any>? = nil,success:((Response) -> ())? = nil,failure:((Error)->())? = nil)  {**

**let ts = Date().milliStamp**

**let nonce = String.randomStr()**

**let bodyJson = JSON(param as Any)**

**let bodyStr = self.jsonToString(json: bodyJson)**

**let sig = (ts + nonce + bodyStr + appSecret).sha1()**

**let requestUrl = BaseUrl + requestUrl  + appid + "/" + ts + "/" + nonce + "/"  + sig + "/"**

**let url = URL(string: requestUrl)!**

**var request = URLRequest(url: url)**

**request.httpMethod = HTTPMethod.post.rawValue**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**let requsetBody = try? JSONSerialization.data(withJSONObject: bodyJson.dictionaryObject ?? [:], options: [])**

**request.httpBody = requsetBody**

**AF.request(request).responseString { response in**

**switch response.result{**

**case .success(\_):**

**if response.response?.statusCode == 200 ,let value = response.value{**

**let json = JSON.init(parseJSON: value )**

**let result = Response(json: json)**

**result.code = response.response?.statusCode ?? 0**

**success?(result)**

**}else{**

**let result = Response(json: JSON.init(parseJSON: response.value ?? ""))**

**result.code = response.response?.statusCode ?? 0**

**success?(result)**

**}**

**break**

**case .failure(let error):**

**failure?(error)**

**break;**

**}**

**}**

**}**

**class func uploadExpireTime() {**

**let exp = UserDefaults.standard.integer(forKey: "product\_tag")**

**let uuid = FCUUID.uuidForDevice() ?? ""**

**MN\_NetworkingTool.requestWithGET(requestUrl: UserExpire, param: ["1":"\(exp)","2":uuid], success: { response in**

**if response.code == 200{**

**}**

**}) { error in**

**}**

**}**

**class func downLoadFile(requestUrl:String,param:Dictionary<String,Any>? = nil,success:((Response) -> ())? = nil,failure:((Error)->())? = nil) {**

**let ts = Date().milliStamp**

**let nonce = String.randomStr()**

**let sig = (ts + nonce + appSecret).sha1()**

**let newparam = param?.sorted(by: { Int($0.key) ?? 0 < Int($1.key) ?? 0**

**})**

**let parStr = newparam?.map({ return "\($0.value)"}).joined(separator: "/") ?? ""**

**let requestUrl = BaseUrl + requestUrl  + parStr + "/" + appid + "/" + ts + "/" + nonce + "/"  + sig  + "/"**

**let url = URL(string: requestUrl)!**

**var request = URLRequest(url: url)**

**request.setValue("application/json", forHTTPHeaderField: "Content-Type")**

**request.httpMethod = "GET"**

**AF.download(requestUrl)**

**.downloadProgress { progress in**

**}**

**.responseData { response in**

**if let data = response.value {**

**let str = parStr + ".mp3"**

**AudioFileManger.writeAudio(data: data, fileName: str) { res in**

**let res = Response(json: "")**

**res.code = 200**

**res.filePath = str**

**success!(res)**

**}**

**}**

**}**

**}**

**class func jsonToString(json:JSON) -> String {**

**var jsonStr = json.rawString() ?? ""**

**jsonStr = jsonStr.replacingOccurrences(of: " ", with: "")**

**jsonStr = jsonStr.replacingOccurrences(of: "\n", with: "")**

**return jsonStr**

**}**

**}**

**@objc class Response: NSObject {**

**init(json:JSON) {**

**self.message = json["msg"].stringValue**

**self.jsonStr = json.rawString() ?? ""**

**self.expires = json["expires"].stringValue**

**self.product\_id = json["product\_id"].stringValue**

**}**

**@objc var message = ""**

**@objc var jsonStr = ""**

**@objc var code = 0**

**@objc var filePath = ""**

**@objc var expires = "-1"**

**@objc var product\_id = ""**

**}**

**import UIKit**

**import Contacts**

**class MN\_UNNotificationTool: AppDelegate,UNUserNotificationCenterDelegate {**

**static let share = MN\_UNNotificationTool()**

**var center:UNUserNotificationCenter?**

**func applicationDidFinishLaunching(\_ application: UIApplication, launchOptions: [UIApplication.LaunchOptionsKey: Any]?) {**

**center = UNUserNotificationCenter.current()**

**center?.delegate = self as UNUserNotificationCenterDelegate**

**center?.getNotificationSettings { (setting) in**

**if setting.authorizationStatus == .notDetermined {**

**self.center?.requestAuthorization(options: [.badge,.sound,.alert]) { (result, error) in**

**if(result){**

**if !(error != nil){**

**}**

**} else{**

**}**

**}**

**} else if (setting.authorizationStatus == .denied){**

**let url = URL(string: UIApplication.openSettingsURLString)**

**if let url = url, UIApplication.shared.canOpenURL(url) {**

**UIApplication.shared.open(url, options: [:],**

**completionHandler: {**

**(success) in**

**})**

**}**

**}else if (setting.authorizationStatus == .authorized){**

**DispatchQueue.main.async {**

**UIApplication.shared.registerForRemoteNotifications()**

**}**

**}else{**

**}**

**}**

**}**

**func getNotifiAuthor() {**

**if #available(iOS 12.0, \*) {**

**UNUserNotificationCenter.current().requestAuthorization(options: [.alert,.badge,.sound,.criticalAlert]) { (granted, error) in**

**if !granted{**

**}**

**}**

**} else {**

**UNUserNotificationCenter.current().requestAuthorization(options: [.alert,.badge,.sound]) { (granted, error) in**

**if !granted{**

**}**

**}**

**}**

**}**

**func createNotificationContent(){**

**MN\_ContactManager.shared.Contacts(allInfo: false) { contacts in**

**var title = userCallNum**

**contacts.forEach { contact in**

**contact.phoneNumbers.forEach { number in**

**if number.value.stringValue == userCallNum{**

**title = contact.fullName()**

**}**

**}**

**}**

**self.addNotifiToCenter(title: title)**

**}**

**}**

**func addNotifiToCenter(title:String) {**

**let content = UNMutableNotificationContent()**

**content.title = title**

**content.subtitle = "Touch to call"**

**content.badge = 0**

**content.categoryIdentifier = "categoryIdentifier"**

**content.sound = UNNotificationSound.default**

**content.launchImageName = "hema.png"**

**let action = UNNotificationAction(identifier: "action", title: "Enter APP", options: UNNotificationActionOptions.foreground)**

**let clearAction = UNNotificationAction(identifier: "clearaction", title: "ignore", options: UNNotificationActionOptions.destructive)**

**let category = UNNotificationCategory(identifier: "categoryIdentifier", actions: [action,clearAction], intentIdentifiers: [], options: [])**

**self.center?.setNotificationCategories([category])**

**let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)**

**let requestidentifier = "requestidentifier"**

**let request = UNNotificationRequest(identifier: requestidentifier, content: content, trigger: trigger)**

**self.center?.add(request) { (error: Error?) in**

**print("成功添加推送")**

**if let tmpError = error {**

**}**

**}**

**}**

**func userNotificationCenter(\_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void) {**

**completionHandler(UNNotificationPresentationOptions.alert)**

**}**

**func userNotificationCenter(\_ center: UNUserNotificationCenter, didReceive response: UNNotificationResponse, withCompletionHandler completionHandler: @escaping () -> Void) {**

**if currentCallType == .centerNumber {**

**NotificationCenter.default.post(Notification(name: Notification.Name(rawValue: "changeCalltype"), object: nil, userInfo: ["calltype":CallNumType.userNumber]))**

**center.removePendingNotificationRequests(withIdentifiers: ["requestidentifier"])**

**center.removeAllDeliveredNotifications()**

**}**

**completionHandler()**

**}**

**}**

**import UIKit**

**import SwiftyStoreKit**

**import StoreKit**

**typealias ApplePayBlock = (Bool) ->()**

**class MN\_PurchaseTool: NSObject {**

**static let sharedInstance: MN\_PurchaseTool = {**

**let instance = MN\_PurchaseTool()**

**return instance**

**}()**

**var success:ApplePayBlock?**

**var fail:ApplePayBlock?**

**func MN\_payWithProduct(productID: String,payResult: ((Bool)->())? = nil) {**

**SVProgressHUD.showWithNativeType()**

**SwiftyStoreKit.purchaseProduct(productID) { result in**

**switch result {**

**case .success(let product):**

**self.MN\_verifyPurchase(forceupdate:true,result: { result in**

**payResult?(result)**

**})**

**if product.needsFinishTransaction {**

**SwiftyStoreKit.finishTransaction(product.transaction)**

**}**

**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)**

**}**

**SVProgressHUD.showError(withStatus: (error as NSError).localizedDescription)**

**}**

**}**

**}**

**func MN\_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) ?? ""**

**MN\_NetworkingTool.verifyReceipt(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{**

**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: "Purchase Success")**

**}**

**}else{**

**result?(false)**

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

**SVProgressHUD.showError(withStatus: "Purchase Failed")**

**}**

**}**

**}else{**

**result?(false)**

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

**SVProgressHUD.showError(withStatus: "Purchase Failed")**

**}**

**}**

**}else{**

**result?(false)**

**SVProgressHUD.showError(withStatus: "Purchase Failed")**

**print("服务器错误")**

**}**

**}) { error in**

**SVProgressHUD.dismiss()**

**}**

**}**

**func updateExpireTime(time:Int) {**

**let defaults = UserDefaults.standard**

**defaults.set(time, forKey: "product\_tag")**

**let now = self.getNowStringMilliStamp()**

**let timeStr = "\(time)"**

**let nowStr = "\(now)"**

**if time > Int(now) ?? 0 {**

**defaults.set(true, forKey: "cur\_status")**

**}**

**MN\_NetworkingTool.uploadExpireTime()**

**}**

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

**SVProgressHUD.showStatus(string: "restoring...")**

**let signal = DispatchSemaphore(value: 0)**

**var isSuccess = false**

**DispatchQueue.global().async {**

**SwiftyStoreKit.restorePurchases(atomically: true) { results in**

**SVProgressHUD.dismiss()**

**if results.restoreFailedPurchases.count > 0 {**

**isSuccess = false**

**signal.signal()**

**SVProgressHUD.showInfo(withStatus: "Restore Faileds")**

**}**

**else if results.restoredPurchases.count > 0 {**

**self.MN\_verifyPurchase(forceupdate: false, isRest: true) { res in**

**if res{**

**isSuccess = true**

**signal.signal()**

**SVProgressHUD.showSuccess(withStatus: "Restore Success")**

**}else{**

**isSuccess = false**

**signal.signal()**

**SVProgressHUD.showInfo(withStatus: "Restore Faileds")**

**}**

**}**

**}**

**else {**

**SVProgressHUD.showInfo(withStatus: "Nothing to Restore")**

**print("Nothing to Restore")**

**isSuccess = false**

**signal.signal()**

**}**

**}**

**signal.wait()**

**if success != nil{**

**success!(isSuccess)**

**}**

**}**

**}**

**func getCurVipStatus() -> Bool{**

**let defaults = UserDefaults.standard**

**let expTime = defaults.integer(forKey: "product\_tag")**

**let now = self.getNowStringMilliStamp()**

**if expTime > Int(now) ?? 0 {**

**defaults.set(true, forKey: "cur\_status")**

**return true**

**}else{**

**defaults.set(false, forKey: "cur\_status")**

**return false**

**}**

**}**

**func getNowStringMilliStamp()->String{**

**let date = Date()**

**let timeInterval:TimeInterval = TimeInterval(date.timeIntervalSince1970)**

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

**return "\(millisecond)"**

**}**

**func goInAppPurchaseView() {**

**}**

**}**

**import UIKit**

**import Contacts**

**import ContactsUI**

**extension String {**

**func contantC() -> Bool {**

**var isContantC = false**

**for char in self.utf8 {**

**if (char > 64 && char < 91) || (char > 96 && char < 123) {**

**isContantC = true**

**}**

**}**

**return isContantC**

**}**

**}**

**extension CNContact {**

**public func PurePhotosForContact() -> (Old:[String], New:[String]){**

**var result\_Old : [String] = []**

**var result\_New : [String] = []**

**let phones = self.phoneNumbers;**

**for phone in phones {**

**result\_Old.append(phone.value.stringValue)**

**let key = MN\_ContactManager.shared.PurePhoneNub(phoneNub: phone)**

**result\_New.append(key)**

**}**

**return (result\_Old, result\_New)**

**}**

**public class func Update(contact : inout CNContact){**

**do {**

**contact = try CNContactStore().unifiedContact(withIdentifier: contact.identifier, keysToFetch: [CNContactViewController.descriptorForRequiredKeys()])**

**}**

**catch { }**

**}**

**public class func UI(contact : inout CNContact, vc : UIViewController, type: Int = 0){**

**if !contact.areKeysAvailable([CNContactViewController.descriptorForRequiredKeys()]) {**

**do {**

**contact = try CNContactStore().unifiedContact(withIdentifier: contact.identifier, keysToFetch: [CNContactViewController.descriptorForRequiredKeys()])**

**}**

**catch { }**

**}**

**var newVC : CNContactViewController**

**if type == 1{**

**newVC = CNContactViewController.init(forUnknownContact: contact)**

**}**

**else if type == 2{**

**newVC = CNContactViewController.init(forNewContact: contact)**

**}else {**

**newVC = CNContactViewController.init(for: contact)**

**}**

**UINavigationBar.appearance().isTranslucent = false**

**UIApplication.shared.delegate?.window??.backgroundColor = UIColor.white**

**vc.navigationController?.pushViewController(newVC, animated: true)**

**}**

**func fullName()->String{**

**if self.familyName.contantC() || self.givenName.contantC() {**

**return self.givenName + self.familyName**

**}else{**

**return self.familyName + self.givenName**

**}**

**}**

**func phoneNubs\_Str() -> String{**

**var res = ""**

**for phone in self.phoneNumbers{**

**let phoneNub  = phone.value.stringValue**

**if res.count > 0{**

**res.append(" . ")**

**}**

**res.append(phoneNub)**

**}**

**return res**

**}**

**func emails\_Str() -> String{**

**var res = ""**

**for email in self.emailAddresses{**

**let emailStr  = email.value as String**

**if res.count > 0{**

**res.append(" . ")**

**}**

**res.append(emailStr)**

**}**

**return res**

**}**

**}**

**class MN\_ContactManager: NSObject {**

**private static let key\_simpleInfo = [**

**CNContactFamilyNameKey,**

**CNContactGivenNameKey,**

**CNContactPhoneNumbersKey,**

**CNContactEmailAddressesKey**

**]**

**private static let key\_allInfo = [**

**CNContactIdentifierKey,**

**CNContactNamePrefixKey,**

**CNContactGivenNameKey,**

**CNContactMiddleNameKey,**

**CNContactFamilyNameKey,**

**CNContactPreviousFamilyNameKey,**

**CNContactNameSuffixKey,**

**CNContactNicknameKey,**

**CNContactOrganizationNameKey,**

**CNContactDepartmentNameKey,**

**CNContactJobTitleKey,**

**CNContactPhoneticGivenNameKey,**

**CNContactPhoneticMiddleNameKey,**

**CNContactPhoneticFamilyNameKey,**

**CNContactPhoneticOrganizationNameKey,**

**CNContactBirthdayKey,**

**CNContactNonGregorianBirthdayKey,**

**CNContactImageDataKey,**

**CNContactThumbnailImageDataKey,**

**CNContactImageDataAvailableKey,**

**CNContactTypeKey,**

**CNContactPhoneNumbersKey,**

**CNContactEmailAddressesKey,**

**CNContactPostalAddressesKey,**

**CNContactDatesKey,**

**CNContactUrlAddressesKey,**

**CNContactRelationsKey,**

**CNContactSocialProfilesKey,**

**CNContactInstantMessageAddressesKey,**

**]**

**public static let shared = MN\_ContactManager();**

**public func GetAthorization(isSuccess:@escaping (Bool)->()){**

**let status = CNContactStore.authorizationStatus(for: .contacts)**

**if status == .notDetermined {**

**let store = CNContactStore()**

**store.requestAccess(for: .contacts) { (granted, error) in**

**if (error != nil) {**

**isSuccess(false)**

**}else{**

**isSuccess(true)**

**}**

**}**

**}else if status == .restricted {**

**self.showAlertViewAboutNotAuthorAccessContact()**

**} else if status == .denied {**

**self.showAlertViewAboutNotAuthorAccessContact()**

**} else if status == .authorized {**

**}**

**isSuccess(true)**

**}**

**func showAlertViewAboutNotAuthorAccessContact() {**

**let alertController: UIAlertController = UIAlertController(title: "Please authorize Contacts permissions", message: "Please allow the APP to access your Contacts in the Settings-Privacy-Address Book option of the iPhone", preferredStyle: .alert)**

**let OKAction: UIAlertAction = UIAlertAction(title: "OK", style: .default, handler: nil)**

**alertController.addAction(OKAction)**

**UIApplication.shared.keyWindow?.rootViewController?.present(alertController, animated: true, completion: nil)**

**}**

**func ReplaceAll(contacts : [CNContact], completed: @escaping ()->()) {**

**self.DeleteAll {**

**for contact in contacts{**

**self.AddContact(contact: contact.mutableCopy() as! CNMutableContact)**

**}**

**completed()**

**}**

**}**

**enum CM\_EmptyInfo\_Type {**

**case name**

**case phone**

**case email**

**}**

**func EmptyInfo(contacts : [CNContact], type: CM\_EmptyInfo\_Type) -> [CNContact] {**

**var result : [CNContact]  = []**

**for contact in contacts {**

**if type == .name &&**

**contact.familyName.count == 0 &&**

**contact.givenName.count == 0 {**

**result.append(contact)**

**}**

**if type == .phone &&**

**contact.phoneNumbers.count == 0 {**

**result.append(contact)**

**}**

**if type == .email && contact.emailAddresses.count == 0{**

**result.append(contact)**

**}**

**}**

**return result**

**}**

**}**

**extension MN\_ContactManager{**

**public func Contacts(allInfo: Bool, completed : @escaping ([CNContact])->()){**

**var result : [CNContact] = []**

**var key = MN\_ContactManager.key\_simpleInfo**

**if allInfo == true {**

**key = MN\_ContactManager.key\_allInfo**

**}**

**let request = CNContactFetchRequest(keysToFetch: key as [CNKeyDescriptor])**

**do {**

**try CNContactStore().enumerateContacts(with: request, usingBlock: { (contact: CNContact, stop: UnsafeMutablePointer<ObjCBool>) in**

**result.append(contact);**

**})**

**} catch {**

**}**

**completed(result)**

**}**

**func Contacts\_sort(completed:@escaping (\_ result :[String : [CNContact]],\_ keys : [String] )->()){**

**var result : [String : [CNContact]] = [:]**

**Contacts(allInfo: true) { (res) in**

**for contact in res{**

**let name = contact.familyName**

**let firstLetterString = MN\_ContactManager.getFirstLetterFromString(aString: name)**

**if result[firstLetterString] != nil{**

**result[firstLetterString]?.append(contact)**

**}else{**

**result[firstLetterString] = [contact]**

**}**

**}**

**var nameKeys = Array(result.keys).sorted()**

**if nameKeys.first == "#" {**

**nameKeys.insert(nameKeys.first!, at: nameKeys.count)**

**nameKeys.remove(at: 0);**

**}**

**completed(result, nameKeys)**

**}**

**}**

**}**

**extension MN\_ContactManager{**

**func AddContacts(contacts: [CNContact]){**

**for contact in contacts{**

**AddContact(contact: contact.mutableCopy() as! CNMutableContact)**

**}**

**}**

**func AddContact(contact: CNMutableContact)  {**

**let saveRequest = CNSaveRequest();**

**saveRequest.add(contact, toContainerWithIdentifier: nil);**

**do {**

**try CNContactStore().execute(saveRequest)**

**}catch{**

**}**

**}**

**func AddContact(name: String, phoneNub: String){**

**AddContact(name, "", phoneNub)**

**}**

**func AddContact(\_ givenName: String, \_ familyName : String, \_ phoneNub: String){**

**AddContact(givenName, familyName, [phoneNub])**

**}**

**func AddContact(\_ givenName: String, \_ familyName : String, \_ phoneNubs: [String]){**

**let contactToAdd = CNMutableContact()**

**contactToAdd.givenName = givenName**

**contactToAdd.familyName = familyName**

**var nubs : [CNLabeledValue<CNPhoneNumber>] = []**

**for phoneNub in phoneNubs {**

**let mobileNumber = CNPhoneNumber(stringValue: phoneNub)**

**let mobileValue = CNLabeledValue(label: CNLabelPhoneNumberMobile, value: mobileNumber)**

**nubs.append(mobileValue)**

**}**

**contactToAdd.phoneNumbers = nubs**

**let saveRequest = CNSaveRequest();**

**saveRequest.add(contactToAdd, toContainerWithIdentifier: nil);**

**do {**

**try CNContactStore().execute(saveRequest)**

**}catch{**

**}**

**}**

**}**

**extension MN\_ContactManager{**

**func Merge(main: CNContact, other: [CNContact]) -> CNMutableContact {**

**var contacts = other;**

**if (contacts.contains(where: { $0 == main })){**

**contacts = contacts.filter{ $0 != main}**

**}**

**var phoneNubs : [CNLabeledValue<CNPhoneNumber>] = []**

**var emails : [CNLabeledValue<NSString>] = []**

**let appendPhoneNumbers = {( contact: CNContact) in**

**for nub\_main in contact.phoneNumbers {**

**let phoneNub\_main = nub\_main.value.stringValue;**

**var needAppend = true;**

**for nub\_cache in phoneNubs{**

**let phoneNub\_cache = nub\_cache.value.stringValue;**

**if (phoneNub\_cache == phoneNub\_main){**

**needAppend = false;**

**continue;**

**}**

**}**

**if (needAppend){**

**phoneNubs.append(nub\_main);**

**}**

**}**

**for email in contact.emailAddresses {**

**let email\_main = email.value as String;**

**var needAppend = true;**

**for email\_cache in emails{**

**if (email\_cache.value as String) == email\_main{**

**needAppend = false;**

**continue;**

**}**

**}**

**if (needAppend){**

**emails.append(email);**

**}**

**}**

**}**

**appendPhoneNumbers(main)**

**for contact in contacts{**

**appendPhoneNumbers(contact);**

**}**

**let mutableContact = main.mutableCopy() as! CNMutableContact**

**mutableContact.phoneNumbers = phoneNubs;**

**mutableContact.emailAddresses = emails**

**return mutableContact;**

**}**

**}**

**extension MN\_ContactManager{**

**func DeleteAll(completed:@escaping()->()) {**

**self.Contacts(allInfo: false) { (res) in**

**self.Delete(contacts: res);**

**completed();**

**}**

**}**

**func Delete(contacts : [CNContact] ) {**

**for contact in contacts {**

**let mutableContact = contact.mutableCopy() as! CNMutableContact;**

**let saveRequest = CNSaveRequest();**

**saveRequest.delete(mutableContact);**

**do {**

**try CNContactStore().execute(saveRequest)**

**}catch{**

**}**

**}**

**}**

**}**

**extension MN\_ContactManager{**

**public func PurePhoneNub(phoneNub : CNLabeledValue<CNPhoneNumber>) -> String{**

**var key = phoneNub.value.stringValue**

**key = key.replacingOccurrences(of: "-", with: " ")**

**key = clearBracket(value: key)**

**key = clearAddSign(value: key)**

**key = key.replacingOccurrences(of: " ", with: "")**

**key = key.replacingOccurrences(of: " ", with: "")**

**return key**

**}**

**func clearBracket(value : String) -> String{**

**if value.contains("(") == false || value.contains(")") == false{**

**return value**

**}**

**return value.components(separatedBy: ")").last ?? value**

**}**

**func clearAddSign(value : String) -> String{**

**let array = value.components(separatedBy: " ")**

**var result = ""**

**for item in array{**

**if item.contains("+"){**

**continue**

**}**

**result += item**

**}**

**return result**

**}**

**func Repeat\_Phone(contacts : [CNContact]) -> [String : [CNContact]] {**

**var result : [String:[CNContact]]  = [:]**

**for contact in contacts {**

**let phones = contact.phoneNumbers;**

**for phone in phones {**

**let key = PurePhoneNub(phoneNub: phone)**

**if var cns = result[key] {**

**if cns.contains(contact) == true{**

**continue**

**}**

**else {**

**cns.append(contact)**

**result[key] = cns**

**}**

**}**

**else {**

**result[key] = [contact]**

**}**

**}**

**}**

**var keys = Array(result.keys)**

**while keys.count > 0 {**

**let key = keys[0]**

**keys.removeFirst()**

**if result[key]?.count == 1 {**

**result.removeValue(forKey: key)**

**}**

**}**

**return result**

**}**

**func Repeat\_Name(contacts : [CNContact]) -> [String : [CNContact]]{**

**var cache : [String : [CNContact] ] = [:]**

**var result : [String : [CNContact]] = [:]**

**for contact in contacts {**

**let key = contact.fullName()**

**if cache.contains(where: { $0.key == key }) {**

**var valuesByKey = cache[key];**

**valuesByKey?.append(contact);**

**cache[key] = valuesByKey;**

**result[key] = valuesByKey;**

**} else {**

**cache[key] = [contact];**

**}**

**}**

**return result;**

**}**

**func Repeat\_Email(contacts : [CNContact]) -> [String : [CNContact]] {**

**var result : [String:[CNContact]]  = [:]**

**for contact in contacts {**

**let emails = contact.emailAddresses;**

**for email in emails {**

**let key = email.value as String**

**if var cns = result[key] {**

**if cns.contains(contact) == true{**

**continue**

**}**

**else {**

**cns.append(contact)**

**result[key] = cns**

**}**

**}**

**else {**

**result[key] = [contact]**

**}**

**}**

**}**

**var keys = Array(result.keys)**

**while keys.count > 0 {**

**let key = keys[0]**

**keys.removeFirst()**

**if result[key]?.count == 1 {**

**result.removeValue(forKey: key)**

**}**

**}**

**return result**

**}**

**}**

**extension MN\_ContactManager{**

**public class func getFirstLetterFromString(aString: String) -> (String) {**

**if aString.count == 0{**

**return "#"**

**}**

**let mutableString = NSMutableString.init(string: aString)**

**CFStringTransform(mutableString as CFMutableString, nil, kCFStringTransformToLatin, false)**

**let pinyinString = mutableString.folding(options: String.CompareOptions.diacriticInsensitive, locale: NSLocale.current)**

**let strPinYin = polyphoneStringHandle(nameString: aString, pinyinString: pinyinString).uppercased()**

**let firstString = strPinYin.substring(to: strPinYin.index(strPinYin.startIndex, offsetBy:1))**

**let regexA = "^[A-Z]$"**

**let predA = NSPredicate.init(format: "SELF MATCHES %@", regexA)**

**return predA.evaluate(with: firstString) ? firstString : "#"**

**}**

**private class func polyphoneStringHandle(nameString:String, pinyinString:String) -> String {**

**if nameString.hasPrefix("长") {return "chang"}**

**if nameString.hasPrefix("沈") {return "shen"}**

**if nameString.hasPrefix("厦") {return "xia"}**

**if nameString.hasPrefix("地") {return "di"}**

**if nameString.hasPrefix("重") {return "chong"}**

**return pinyinString;**

**}**

**}**

**import UIKit**

**import AVFoundation**

**class AudioPlayer: NSObject {**

**var player:AVAudioPlayer!**

**init(url:String,success:((Bool)->())? = nil) {**

**let path = NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).first ?? ""**

**player = try? AVAudioPlayer(contentsOf: URL(string: path + "/\(url)")!)**

**}**

**func stop(){**

**player?.stop()**

**}**

**func pause() {**

**player?.pause()**

**}**

**func play()->Bool{**

**if (player?.isPlaying)! {**

**player?.pause()**

**return false**

**}else{**

**player?.play()**

**return true**

**}**

**}**

**func progress()->Double{**

**return (player?.currentTime)!/(player?.duration)!**

**}**

**func currentTime()->Double{**

**return (player?.currentTime)!**

**}**

**func isPlaying()->Bool{**

**return (player?.isPlaying ?? false)!**

**}**

**}**

**import UIKit**

**class FMDBManger: NSObject {**

**private static let manger:FMDBManger = FMDBManger()**

**class func shareManger()->FMDBManger{**

**return manger**

**}**

**var db:FMDatabase?**

**func openDB(DBName:String) {**

**let path = (NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).last ?? "") + "/\(DBName)"**

**db = FMDatabase(path: path)**

**if !db!.open() {**

**return**

**}**

**createTable()**

**}**

**func createTable() {**

**let sql = """**

**CREATE TABLE IF NOT EXISTS Record(id INTEGER PRIMARY KEY AUTOINCREMENT,**

**recording\_uuid TEXT,**

**recording\_url TEXT,**

**\_id TEXT,**

**record\_url TEXT,**

**start\_time TEXT,**

**size INTEGER,**

**conversation\_uuid TEXT,**

**timestamp TEXT,**

**end\_time TEXT,**

**user\_uuid TEXT,**

**record\_name TEXT,**

**local\_url TEXT**

**);**

**"""**

**if db!.executeUpdate(sql, withArgumentsIn: []) {**

**}else{**

**}**

**}**

**func insertRecord(record:RecordModel) {**

**let sql = """**

**INSERT INTO Record(recording\_uuid,**

**recording\_url,**

**\_id,**

**record\_url,**

**start\_time,**

**size,**

**conversation\_uuid,**

**timestamp,**

**end\_time,**

**user\_uuid,**

**record\_name,**

**local\_url**

**)**

**SELECT '\(record.recording\_uuid)',**

**'\(record.recording\_url)',**

**'\(record.record\_id)',**

**'\(record.record\_url)',**

**'\(record.start\_time)',**

**'\(record.size)',**

**'\(record.conversation\_uuid)',**

**'\(record.timestamp)',**

**'\(record.end\_time)',**

**'\(record.user\_uuid)',**

**'\(record.record\_name)',**

**'\(record.local\_url)'**

**WHERE NOT EXISTS(SELECT \_id FROM Record WHERE \_id = '\(record.record\_id)')**

**"""**

**if db!.open() {**

**if db!.executeUpdate(sql, withArgumentsIn: []) {**

**}else{**

**}**

**}**

**db!.close()**

**}**

**func selectAllRecord()->Array<RecordModel>{**

**let sql = """**

**SELECT \* FROM Record**

**"""**

**var resArr:Array<RecordModel> = []**

**if db!.open() {**

**do {**

**let res =  try db!.executeQuery(sql, values: [])**

**while res.next() {**

**let model = RecordModel(res: res)**

**resArr.append(model)**

**}**

**} catch  {**

**}**

**}**

**db!.close()**

**return resArr**

**}**

**func deleteAllRecord()->Array<RecordModel>{**

**let sql = """**

**DELETE FROM Record WHERE 1 = 1 ''**

**"""**

**var resArr:Array<RecordModel> = []**

**if db!.open() {**

**do {**

**let res =  try db!.executeQuery(sql, values: [])**

**while res.next() {**

**let model = RecordModel(res: res)**

**resArr.append(model)**

**}**

**} catch  {**

**}**

**}**

**db!.close()**

**return resArr**

**}**

**func deleteRecord(recordId:String) {**

**let sql = """**

**DELETE FROM Record WHERE \_id = '\(recordId)'**

**"""**

**if db!.open() {**

**if db!.executeUpdate(sql, withArgumentsIn: []) {**

**}else{**

**}**

**}**

**db!.close()**

**}**

**func searchRecordLocalurl(recording\_uuid:String) ->String {**

**let sql = """**

**SELECT \* FROM Record WHERE recording\_uuid = '\(recording\_uuid)'**

**"""**

**var resArr:Array<RecordModel> = []**

**if db!.open() {**

**do {**

**let res =  try db!.executeQuery(sql, values: [])**

**while res.next() {**

**let model = RecordModel(res: res)**

**resArr.append(model)**

**}**

**} catch  {**

**}**

**}**

**db!.close()**

**return resArr.first?.local\_url ?? ""**

**}**

**func changeRecordName(recording\_uuid:String,newName:String) {**

**let sql = """**

**UPDATE Record SET record\_name = '\(newName)' WHERE recording\_uuid = '\(recording\_uuid)'**

**"""**

**if db!.open() {**

**if db!.executeUpdate(sql, withArgumentsIn: []) {**

**}else{**

**}**

**}**

**db!.close()**

**}**

**func addLocalurl(recording\_uuid:String,localurl:String) {**

**let sql = """**

**UPDATE Record SET local\_url = '\(localurl)' WHERE recording\_uuid = '\(recording\_uuid)' AND local\_url = ''**

**"""**

**if db!.open() {**

**if db!.executeUpdate(sql, withArgumentsIn: []) {**

**}else{**

**}**

**}**

**db!.close()**

**}**

**}**

**import UIKit**

**class AudioFileManger: NSObject {**

**static let manger = FileManager.default**

**static let filePath = NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).first ?? ""**

**class func writeAudio(data:Data,fileName:String,success:(Bool)->()){**

**let path = AudioFileManger.filePath + "/\(fileName)"**

**do {**

**try data.write(to: URL(fileURLWithPath: path))**

**success(true)**

**} catch  {**

**}**

**}**

**class func deleteFile(fileName:String) {**

**manger.changeCurrentDirectoryPath(filePath)**

**do {**

**try manger.removeItem(atPath: fileName)**

**} catch  {**

**}**

**}**

**}**

**import UIKit**

**import SwiftyStoreKit**

**import SwiftyJSON**

**import Alamofire**

**@UIApplicationMain**

**class AppDelegate: UIResponder, UIApplicationDelegate {**

**var window: UIWindow?**

**let manager = NetworkReachabilityManager(host: "http://www.baidu.com")**

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

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

**for purchase in purchases {**

**switch purchase.transaction.transactionState {**

**case .purchased, .restored:**

**if purchase.needsFinishTransaction {**

**SwiftyStoreKit.finishTransaction(purchase.transaction)**

**}**

**case .failed, .purchasing, .deferred:**

**break**

**default:**

**break;**

**}**

**}**

**}**

**MN\_CallStatusTool.share.applicationDidFinishLaunching(application)**

**MN\_UNNotificationTool.share.applicationDidFinishLaunching(application, launchOptions: launchOptions)**

**NotificationCenter.default.addObserver(self, selector: #selector(changeRootView), name: NSNotification.Name("changeRoot"), object: nil)**

**window = UIWindow.init(frame: UIScreen.main.bounds)**

**window?.makeKeyAndVisible()**

**if !UserDefaults.standard.bool(forKey: "isfirst") {**

**UserDefaults.standard.set(true, forKey: "isfirst")**

**let vc = MN\_GuideViewController.loadStoryboard(name: "Main")**

**self.window?.rootViewController = vc**

**}else{**

**let tab = BaseTabBarController.loadStoryboard(name: "Main")**

**self.window?.rootViewController = tab**

**}**

**mn\_getCenterNumber()**

**mn\_userExit()**

**mn\_updateUser()**

**NotificationCenter.default.addObserver(self, selector: #selector(changeCallNumType(noti:)), name: NSNotification.Name(rawValue: "changeCalltype"), object: nil)**

**NotificationCenter.default.addObserver(self, selector: #selector(mn\_netChange), name: NSNotification.Name(rawValue: "NetworkStatusChangedNotiName"), object: nil)**

**manager?.startListening(onUpdatePerforming: { (status) in**

**switch status{**

**case .unknown:**

**break**

**case .notReachable:**

**break**

**case .reachable(.cellular),.reachable(.ethernetOrWiFi):**

**NotificationCenter.default.post(name: NSNotification.Name("NetworkStatusChangedNotiName"), object: nil)**

**}**

**})**

**return true**

**}**

**@objc func mn\_netChange()  {**

**self.mn\_getCenterNumber()**

**self.mn\_userExit()**

**self.mn\_updateUser()**

**}**

**func mn\_updateUser() {**

**let uuid = FCUUID.uuidForDevice() ?? ""**

**MN\_NetworkingTool.requestWithPOST(requestUrl: UpdateUser, param: ["uuid":uuid], success: { result  in**

**if result.code == 200{**

**}**

**}) { error in**

**}**

**}**

**func mn\_userExit() {**

**let uuid = FCUUID.uuidForDevice() ?? ""**

**if !UserDefaults.standard.bool(forKey: "islogin") {**

**MN\_NetworkingTool.requestWithGET(requestUrl: UserExit,param: ["uuid":uuid], success: { result in**

**if result.code == 403{**

**UserDefaults.standard.set(false, forKey: "islogin")**

**}else if result.code == 200 {**

**UserDefaults.standard.set(true, forKey: "islogin")**

**}**

**}) { error in**

**}**

**}**

**}**

**func mn\_getCenterNumber() {**

**let identifier = NSLocale.current.identifier**

**let locationId = NSLocale(localeIdentifier: identifier)**

**let countryLoc = locationId.object(forKey: .countryCode) as? String ?? ""**

**let uuid = FCUUID.uuidForDevice() ?? ""**

**MN\_NetworkingTool.requestWithPOST(requestUrl: CenterNum, param: ["country\_code":countryLoc.uppercased(),"uuid":uuid], success: { response in**

**if response.code == 200{**

**centerNum = "+" + (JSON(parseJSON: response.jsonStr).dictionaryObject?["number"] as? String ?? "")**

**}**

**}) { error in**

**}**

**}**

**@objc func changeCallNumType(noti:Notification){**

**currentCallType = noti.userInfo?["calltype"] as? CallNumType ?? .userNumber**

**if userCallNum == "" {**

**return**

**}**

**let phone = "telprompt://" + userCallNum**

**if UIApplication.shared.canOpenURL(URL(string: phone)!) {**

**UIApplication.shared.open(URL(string: phone)!, options: [:],**

**completionHandler: {**

**(success) in**

**})**

**}**

**}**

**@objc func changeRootView(){**

**if ((window?.rootViewController) != nil){**

**if window?.rootViewController?.view.subviews.count != 0 {**

**guard let views =  window?.rootViewController?.view.subviews else {**

**return**

**}**

**for view in views {**

**view.removeFromSuperview()**

**}**

**}**

**window?.rootViewController = nil**

**}**

**let tab = BaseTabBarController.loadStoryboard(name: "Main")**

**self.window?.rootViewController = tab**

**}**

**func applicationWillEnterForeground(\_ application: UIApplication) {**

**PTEventRecord.shareManager().addEvent(withType: "launch", name: "app", extras: EventLog(message: "launch"))**

**}**

**func applicationDidEnterBackground(\_ application: UIApplication) {**

**PTEventRecord.shareManager().addEvent(withType: "backgroud", name: "app", extras: EventLog(message: "backgroud"))**

**}**

**}**

**import UIKit**

**class MN\_GuideScrollView: UIView ,NibLoadable,UIScrollViewDelegate{**

**@IBOutlet weak var scrollView: UIScrollView!**

**@IBOutlet weak var pageController: UIPageControl!**

**@IBOutlet var imageViews: [UIImageView]!**

**@IBOutlet var titleLab: [UILabel]!**

**@IBOutlet var subLabs: [UILabel]!**

**@IBOutlet var restoreBtns: [UIButton]!**

**var images: Array<String> = [] {**

**didSet{**

**imageViews.forEach { imageView in**

**imageView.image = UIImage(named: images[imageView.tag])**

**}**

**}**

**}**

**var titles: Array<String> = [] {**

**didSet{**

**titleLab.forEach { lab in**

**lab.text = titles[lab.tag]**

**}**

**}**

**}**

**var subTitles: Array<String> = [] {**

**didSet{**

**subLabs.forEach { lab in**

**lab.text = subTitles[lab.tag]**

**}**

**}**

**}**

**func setRestoreHidden() {**

**for view in restoreBtns {**

**view.isHidden = true**

**}**

**}**

**func scrollViewDidScroll(\_ scrollView: UIScrollView) {**

**let index = Int(scrollView.contentOffset.x / LJScreenWidth)**

**pageController.currentPage = index**

**PTEventRecord.shareManager().addEvent(withType: "scroll", name: "scroll\_index", extras: EventLog(message: "\(index)"))**

**}**

**@IBAction func restoreClick(\_ sender: UIButton) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_restore", extras: EventLog(message: ""))**

**MN\_PurchaseTool.sharedInstance.restorePurchases { bool in**

**}**

**}**

**}**

**import UIKit**

**class MN\_GuideViewController: UIViewController,StoryboardLoadable,UIScrollViewDelegate {**

**lazy var scrollView: MN\_GuideScrollView = {**

**let scroll = MN\_GuideScrollView.loadFromNib()**

**scroll.frame = CGRect(x: 0, y: MN\_StatusBar\_Height, width: LJScreenWidth, height: LJScreenHeight - MN\_StatusBar\_Height - MN\_TabbarSafeBottomMargin - 130)**

**return scroll**

**}()**

**var mainScrollerView: UIScrollView!**

**@IBOutlet var guideBottom: UIView!**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**let scroll = UIScrollView(frame: CGRect(x: 0, y: 0, width: LJScreenWidth, height: LJScreenHeight))**

**scroll.contentSize = CGSize(width: LJScreenWidth, height: LJScreenHeight)**

**scroll.delegate = self**

**self.view.addSubview(scroll)**

**scroll.addSubview(self.scrollView)**

**mainScrollerView = scroll**

**MN\_makeBottomView()**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**func MN\_makeBottomView(){**

**guideBottom.frame = CGRect(x: 0, y: self.scrollView.frame.origin.y + self.scrollView.frame.size.height , width: LJScreenWidth, height: 130 + MN\_TabbarSafeBottomMargin )**

**mainScrollerView.addSubview(guideBottom)**

**}**

**@IBAction func startRecord(\_ sender: UITapGestureRecognizer) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "startRecord", extras: EventLog(message: "enter"))**

**let exit = UserDefaults.standard.bool(forKey: "islogin")**

**if !exit {**

**let vc = MN\_PurchaseViewController.loadStoryboard(name: "Main")**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}else{**

**NotificationCenter.default.post(Notification(name: Notification.Name("changeRoot")))**

**}**

**}**

**@IBAction func privacyClick(\_ sender: Any) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_privacy", extras: EventLog(message: "priviacy"))**

**let vc = MN\_PureWebViewController()**

**vc.urlStr = "https://www.callsnotice.com/call/privacy.html"**

**vc.titleStr = "Privacy Policy"**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}**

**@IBAction func termsClick(\_ sender: Any) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_terms", extras: EventLog(message: "terms"))**

**let vc = MN\_PureWebViewController()**

**vc.urlStr = "https://www.callsnotice.com/call/terms.html"**

**vc.titleStr = "Terms of Use";**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}**

**}**

**import UIKit**

**import SwiftyJSON**

**typealias disMissBlock = ()->()**

**class MN\_RegisterViewController: UIViewController ,StoryboardLoadable,CodeDelegate{**

**@IBOutlet weak var areaLab: UILabel!**

**@IBOutlet weak var interCodeLab: UILabel!**

**@IBOutlet weak var phoneTF: UITextField!**

**@IBOutlet weak var flagImage: UIImageView!**

**var backBtn:UIButton?**

**var currentModel:CoutryModel?{**

**didSet{**

**interCodeLab.text = "+" + (currentModel?.Dial ?? "1")**

**areaLab.text = currentModel?.CLDR\_display\_name**

**flagImage.image = UIImage(named: "MN\_" + (currentModel?.ISO3166\_1\_Alpha\_2 ?? "US"))**

**}**

**}**

**var countryModelArr = MN\_CountryManager.MN\_getCountryModels()**

**var block:disMissBlock?**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**let identifier = NSLocale.current.identifier**

**let locationId = NSLocale(localeIdentifier: identifier)**

**let countryLoc = locationId.object(forKey: .countryCode) as? String ?? ""**

**if let model = countryModelArr.first(where: {$0.ISO3166\_1\_Alpha\_2 == countryLoc}){**

**currentModel = model**

**}**

**}**

**@objc func disMiss() {**

**self.dismiss(animated: true, completion: nil)**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**let button = UIButton(frame: CGRect(x: 0, y: 0, width: 44, height: 44))**

**button.addTarget(self, action: #selector(disMiss), for: .touchUpInside)**

**button.setImage(UIImage(named: "navback"), for: .normal)**

**button.setImage(UIImage(named: "navback"), for: .selected)**

**self.backBtn = button**

**self.navigationController?.navigationBar.addSubview(self.backBtn!)**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**self.backBtn?.removeFromSuperview()**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**@IBAction func countryTap(\_ sender: UITapGestureRecognizer) {**

**let vc = MN\_CountryListViewController.loadStoryboard(name: "Main")**

**vc.block = { model in**

**self.currentModel = model**

**}**

**self.present(vc, animated: true, completion: nil)**

**}**

**@IBAction func ContinueClick(\_ sender: UIButton) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_continue", extras: EventLog(message: ""))**

**if self.phoneTF.text == "" {**

**SVProgressHUD.showInfo(string: "Please enter your phone number")**

**return**

**}**

**let countryCode = interCodeLab.text ?? ""**

**let phoneNum = self.phoneTF.text ?? ""**

**let sheet = UIAlertController(title: "Are you sure?", message: "\(countryCode)\(phoneNum) Only calls with this mobile number can be recorded", preferredStyle: .alert)**

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

**MN\_NetworkingTool.requestWithPOST(requestUrl: UpdateUser, param: ["number":countryCode+phoneNum,"uuid":FCUUID.uuidForDevice() ?? ""]) { result in**

**if result.code == 403{**

**UserDefaults.standard.set(false, forKey: "islogin")**

**}else if result.code == 200 {**

**UserDefaults.standard.set(true, forKey: "islogin")**

**self.dismiss(animated: true, completion: nil)**

**}**

**} failure: { error in**

**}**

**}**

**let cancel = UIAlertAction(title: "Cancel", style: .cancel) { action in**

**}**

**sheet.addAction(more)**

**sheet.addAction(cancel)**

**self.present(sheet, animated: true, completion: nil)**

**}**

**func viewDissMiss() {**

**self.dismiss(animated: false, completion: {**

**weak var weakSelf = self**

**weakSelf?.block?()**

**})**

**}**

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

**self.phoneTF.resignFirstResponder()**

**}**

**deinit {**

**LJPrint("deinit")**

**}**

**}**

**import UIKit**

**typealias SelectBlock = ( \_ model:CoutryModel)->()**

**class MN\_CountryListViewController: UIViewController,UITableViewDelegate,UITableViewDataSource,StoryboardLoadable {**

**@IBOutlet weak var tableView: UITableView!**

**var keys:Array<String> = []**

**var countryArrayDic:Dictionary<String,Array<CoutryModel>> = [:]**

**var block:SelectBlock?**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**countryArrayDic = MN\_CountryManager.MN\_getCountryCategoryList()**

**keys = countryArrayDic.keys.sorted(by: < )**

**}**

**@IBAction func cancelClick(\_ sender: UIButton) {**

**self.dismiss(animated: true, completion: nil)**

**}**

**func numberOfSections(in tableView: UITableView) -> Int {**

**return keys.count**

**}**

**func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {**

**return countryArrayDic[keys[section]]?.count ?? 0**

**}**

**func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {**

**guard let model = countryArrayDic[keys[indexPath.section]]?[indexPath.row] else{**

**return UITableViewCell()**

**}**

**let cell = tableView.dequeueReusableCell(withIdentifier: "countryCell")**

**(cell?.viewWithTag(100) as? UIImageView)?.image = UIImage(named: "MN\_" + model.ISO3166\_1\_Alpha\_2 + ".png")**

**(cell?.viewWithTag(101) as? UILabel)?.text = model.CLDR\_display\_name**

**(cell?.viewWithTag(102) as? UILabel)?.text = "+" + model.Dial**

**return cell ?? UITableViewCell()**

**}**

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

**return 28**

**}**

**func tableView(\_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {**

**let view = UIView(frame: CGRect(x: 0, y: 0, width: LJScreenWidth, height: 28))**

**view.backgroundColor = UIColor.colorWithHexColorString("F8F8F8", alpha: 0.82)**

**let lab = UILabel(frame: CGRect(x: 16, y: 0, width: 100, height: 28))**

**lab.text = keys[section]**

**lab.font = UIFont.pingFangSC\_Semibold(size: 17)**

**view.addSubview(lab)**

**return view**

**}**

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

**guard let model = countryArrayDic[keys[indexPath.section]]?[indexPath.row] else{**

**self.dismiss(animated: true, completion: nil)**

**return**

**}**

**block?(model)**

**self.dismiss(animated: true, completion: nil)**

**}**

**func sectionIndexTitles(for tableView: UITableView) -> [String]? {**

**return keys**

**}**

**func tableView(\_ tableView: UITableView, sectionForSectionIndexTitle title: String, at index: Int) -> Int {**

**return index**

**}**

**}**

**import UIKit**

**import SnapKit**

**import SwiftyJSON**

**protocol CodeDelegate {**

**func viewDissMiss()**

**}**

**class MN\_CodeViewController: UIViewController,UITextFieldDelegate,CodeTextFieldDelegate, VerificationCodeViewDelegate,StoryboardLoadable {**

**var phoneNum :String = ""**

**var interCode : String = ""**

**var delegate:CodeDelegate?**

**var request\_id = ""**

**var aVerificationCodeView:VerificationCodeView?**

**@IBOutlet weak var resendBtn: UIButton!**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**aVerificationCodeView = VerificationCodeView(value: "", phoneNumber: phoneNum, requestId: request\_id)**

**aVerificationCodeView?.delegate = self**

**aVerificationCodeView?.requestId = self.request\_id**

**self.view.addSubview(aVerificationCodeView!)**

**aVerificationCodeView?.snp.makeConstraints { make in**

**make.left.right.equalToSuperview()**

**make.top.equalTo(self.view.snp\_topMargin).offset(110)**

**make.height.equalTo(aVerificationCodeView!.itemwidth)**

**}**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**setTimeBtn()**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**var codeTimer:DispatchSourceTimer?**

**func setTimeBtn() {**

**resendBtn.isEnabled = false**

**var remainingCount: Int = 0 {**

**willSet {**

**if newValue > 180 {**

**let att = NSMutableAttributedString(string: "Resend Code")**

**att.addAttributes([NSAttributedString.Key.font : UIFont.pingFangSC\_Semibold(size: 16),NSAttributedString.Key.foregroundColor : UIColor.colorWithHexColorString("2EC849", alpha: 1)], range: NSRange(location: 0, length: att.length))**

**resendBtn.setAttributedTitle(att, for: .normal)**

**resendBtn.isEnabled = true**

**}else{**

**let att = NSMutableAttributedString(string: formatTime(time: newValue))**

**att.addAttributes([NSAttributedString.Key.font : UIFont.pingFangSC\_Semibold(size: 16),NSAttributedString.Key.foregroundColor : UIColor.colorWithHexColorString("000000", alpha: 0.3)], range: NSRange(location: 0, length: att.length))**

**att.addAttributes([NSAttributedString.Key.foregroundColor : UIColor.colorWithHexColorString("000000", alpha: 0.7)], range: NSRange(location: 0, length: 6))**

**resendBtn.setAttributedTitle(att, for: .normal)**

**}**

**}**

**}**

**codeTimer = DispatchSource.makeTimerSource(queue:DispatchQueue.global())**

**codeTimer?.schedule(deadline: .now(), repeating: .seconds(1))**

**codeTimer?.setEventHandler(handler: {**

**DispatchQueue.main.async {**

**remainingCount += 1**

**if remainingCount > 180 {**

**self.codeTimer?.cancel()**

**}**

**}**

**})**

**codeTimer?.resume()**

**}**

**func formatTime(time:Int) -> String {**

**let min = time/60**

**let sec = time%60**

**var minStr = ""**

**var secStr = ""**

**if min < 10 {**

**minStr = "0" + "\(min)"**

**}else{**

**minStr = "\(min)"**

**}**

**if sec < 10 {**

**secStr = "0" + "\(sec)"**

**}else{**

**secStr = "\(sec)"**

**}**

**return "\(minStr):\(secStr)/03:00"**

**}**

**@IBAction func reSendBtn(\_ sender: UIButton) {**

**MN\_NetworkingTool.requestWithPOST(requestUrl: SendCode, param: ["number":interCode+phoneNum,"msg":"CallRecorder","uuid":FCUUID.uuidForDevice() ?? ""], success: { result in**

**if result.code == 200{**

**self.request\_id = JSON(parseJSON: result.jsonStr).dictionaryObject?["request\_id"] as! String**

**self.aVerificationCodeView?.requestId = self.request\_id**

**}**

**self.setTimeBtn()**

**}) { error in**

**}**

**}**

**func didClickBackWard() {**

**}**

**func verificationCodeView(succcess: Bool) {**

**if succcess {**

**let uuid = FCUUID.uuidForDevice() ?? ""**

**MN\_NetworkingTool.requestWithPOST(requestUrl: UpdateUser, param: ["number":interCode+phoneNum,"uuid":uuid], success: { result in**

**if result.code == 200{**

**self.delegate?.viewDissMiss()**

**self.codeTimer?.cancel()**

**NotificationCenter.default.post(Notification(name: Notification.Name("changeRoot")))**

**}else{**

**SVProgressHUD.showInfo(string: "error")**

**}**

**}) { error in**

**}**

**}else{**

**SVProgressHUD.showInfo(string: "code error")**

**self.aVerificationCodeView?.cleanVerificationCodeView()**

**}**

**}**

**}**

**protocol CodeTextFieldDelegate {**

**func didClickBackWard()**

**}**

**class CodeTextField: UITextField {**

**var deleteDelegate:CodeTextFieldDelegate?**

**override func deleteBackward() {**

**super.deleteBackward()**

**deleteDelegate?.didClickBackWard()**

**}**

**}**

**protocol VerificationCodeViewDelegate {**

**func verificationCodeView(succcess:Bool)**

**}**

**class VerificationCodeView: UIView,UITextFieldDelegate,CodeTextFieldDelegate {**

**var delegate:VerificationCodeViewDelegate?**

**var value = ""**

**var phoneNumber = ""**

**var requestId = ""**

**var codes:[String]{**

**get {**

**return Array(value).map({$0.description})**

**}**

**}**

**var textFields:[UITextField] = []**

**var numOfRect:Int{**

**get{**

**return codes.count**

**}**

**}**

**let space : CGFloat = 16**

**var itemwidth : CGFloat = 0**

**var itemheight : CGFloat = 0**

**init(value:String,phoneNumber:String,requestId:String) {**

**super.init(frame: CGRect.zero)**

**self.phoneNumber = phoneNumber**

**if value == "" {**

**self.value = "      "**

**}else {**

**self.value = value**

**}**

**self.requestId = requestId**

**self.isUserInteractionEnabled = false**

**itemwidth = (LJScreenWidth - space \* 7 ) / 6**

**itemheight = itemwidth \* 48 / 42**

**self.codes.enumerated().forEach { index,str in**

**let tv = CodeTextField(frame: CGRect(x: (itemwidth + space) \* CGFloat(index) + space, y: 0, width: itemwidth, height: itemheight))**

**tv.tag = index**

**tv.placeholder = str**

**tv.textAlignment = .center**

**tv.font = UIFont.boldSystemFont(ofSize: 30)**

**tv.textColor = UIColor.init(red: 51/255, green: 51/255, blue: 51/255, alpha: 1)**

**tv.keyboardType = .phonePad**

**tv.backgroundColor = .white**

**tv.delegate = self**

**tv.deleteDelegate = self**

**addSubview(tv)**

**tv.layer.cornerRadius = 12**

**tv.layer.borderWidth = 1**

**tv.layer.borderColor = UIColor.colorWithHexColorString("000000", alpha: 0.1).cgColor**

**textFields.append(tv)**

**}**

**textFields.first?.becomeFirstResponder()**

**}**

**required init?(coder aDecoder: NSCoder) {**

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

**}**

**func isPurnInt(string: String) -> Bool {**

**let scan: Scanner = Scanner(string: string)**

**var val:Int = 0**

**return scan.scanInt(&val) && scan.isAtEnd**

**}**

**func textField(\_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString string: String) -> Bool {**

**if !isPurnInt(string: string) {**

**return false**

**}**

**if !textField.hasText {**

**let index = textField.tag**

**textField.resignFirstResponder()**

**if index == numOfRect - 1 {**

**textFields[index].text = string**

**var code = ""**

**for tv in textFields{**

**code += tv.text ?? ""**

**}**

**verificationCode(code: code) { bool in**

**self.delegate?.verificationCodeView(succcess: bool)**

**}**

**return false**

**}**

**textFields[index].text = string**

**textFields[index + 1].becomeFirstResponder()**

**}**

**return false**

**}**

**func verificationCode(code:String,success:((Bool)->())? = nil) {**

**if requestId == "" {**

**return**

**}**

**MN\_NetworkingTool.requestWithPOST(requestUrl: VerifyCode, param: ["code":code,"request\_id":requestId,"uuid":FCUUID.uuidForDevice() ?? ""], success: { response in**

**if response.code == 200{**

**if JSON(parseJSON: response.jsonStr)["status"] == "ok" {**

**UserDefaults.standard.set(true, forKey: "islogin")**

**success?(true)**

**}else{**

**success?(false)**

**}**

**}else if response.code == 403{**

**success?(false)**

**}**

**}) { error in**

**}**

**}**

**func didClickBackWard() {**

**for i in 1..<numOfRect{**

**if !textFields[i].isFirstResponder {**

**continue**

**}**

**textFields[i].resignFirstResponder()**

**textFields[i-1].becomeFirstResponder()**

**textFields[i-1].text = ""**

**}**

**}**

**func cleanVerificationCodeView(){**

**textFields.forEach({$0.text = ""})**

**textFields.first?.becomeFirstResponder()**

**}**

**}**

**import UIKit**

**enum PurchaseProducts:String {**

**case week = "0"**

**case month = "1"**

**case year = "2"**

**case noproduct = ""**

**}**

**let weekProId = "co.phone.recorder.week"**

**let monthProId = "co.phone.recorder.month"**

**let yearProid = "co.phone.recorder.year"**

**class MN\_PurchaseViewController: UIViewController,StoryboardLoadable,UITextViewDelegate {**

**@IBOutlet weak var inBtnLabel: UILabel!**

**@IBOutlet var productViews: [UIView]!**

**@IBOutlet weak var scrollView: UIScrollView!**

**@IBOutlet weak var textView: UITextView!**

**var currentProduct:PurchaseProducts = .noproduct{**

**didSet{**

**for view in productViews {**

**view.layer.borderWidth = 0.5**

**view.layer.borderColor = UIColor.colorWithHexColorString("000000", alpha: 0.1).cgColor**

**if currentProduct == .week && view.tag == 0{**

**view.layer.borderWidth = 4**

**view.layer.borderColor = UIColor.colorWithHexColorString("2EC849", alpha: 1).cgColor**

**}**

**if currentProduct == .month && view.tag == 1{**

**view.layer.borderWidth = 4**

**view.layer.borderColor = UIColor.colorWithHexColorString("2EC849", alpha: 1).cgColor**

**}**

**if currentProduct == .year && view.tag == 2{**

**view.layer.borderWidth = 4**

**view.layer.borderColor = UIColor.colorWithHexColorString("2EC849", alpha: 1).cgColor**

**}**

**}**

**}**

**}**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**setTextView()**

**for view in productViews{**

**view.layer.borderWidth = 0.5**

**view.layer.borderColor = UIColor.colorWithHexColorString("000000", alpha: 0.1).cgColor**

**weekClick(UITapGestureRecognizer())**

**}**

**scrollView.scrollsToTop = true**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**}**

**override func viewDidAppear(\_ animated: Bool) {**

**}**

**@IBAction func weekClick(\_ sender: UITapGestureRecognizer) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_Purchaseweek", extras: EventLog(message: "weekclick"))**

**currentProduct = .week**

**inBtnLabel.text = "Skip the free trial and subscribe"**

**}**

**@IBAction func monthClick(\_ sender: UITapGestureRecognizer) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_Purchasemonth", extras: EventLog(message: "monthclick"))**

**currentProduct = .month**

**inBtnLabel.text = "Come and get 3 days for free"**

**}**

**@IBAction func yearClick(\_ sender: UITapGestureRecognizer) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_Purchaseyear", extras: EventLog(message: "yearclick"))**

**currentProduct = .year**

**inBtnLabel.text = "Come and get 3 days for free"**

**}**

**@IBAction func PurchaseClick(\_ sender: UITapGestureRecognizer) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_Purchase", extras: EventLog(message: "Purchase"))**

**var proid = ""**

**switch currentProduct {**

**case .week:**

**proid = weekProId**

**break**

**case .month:**

**proid = monthProId**

**break**

**case .year:**

**proid = yearProid**

**break**

**default:**

**break**

**}**

**MN\_PurchaseTool.sharedInstance.MN\_payWithProduct(productID: proid) { res in**

**if res{**

**DispatchQueue.main.async {**

**if !UserDefaults.standard.bool(forKey: "islogin") {**

**let vc = MN\_RegisterViewController.loadStoryboard(name: "Main")**

**vc.block = {**

**self.dismiss(animated: false, completion: nil)**

**}**

**let nav = BaseNavigationController(rootViewController: vc)**

**nav.modalPresentationStyle = .fullScreen**

**self.present(nav, animated: true, completion: nil)**

**}else{**

**self.dismiss(animated: true, completion: nil)**

**NotificationCenter.default.post(Notification(name: Notification.Name("changeRoot")))**

**}**

**}**

**}**

**}**

**}**

**@IBAction func restoreClick(\_ sender: UIButton) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_restore", extras: EventLog(message: "restore"))**

**MN\_PurchaseTool.sharedInstance.restorePurchases { res in**

**if res{**

**if !UserDefaults.standard.bool(forKey: "islogin") {**

**DispatchQueue.main.async {**

**let vc = MN\_RegisterViewController.loadStoryboard(name: "Main")**

**vc.block = {**

**self.dismiss(animated: false, completion: nil)**

**}**

**let nav = BaseNavigationController(rootViewController: vc)**

**nav.modalPresentationStyle = .fullScreen**

**self.present(nav, animated: true, completion: nil)**

**}**

**}else{**

**DispatchQueue.main.async {**

**self.dismiss(animated: true, completion: nil)**

**NotificationCenter.default.post(Notification(name: Notification.Name("changeRoot")))**

**}**

**}**

**}**

**}**

**}**

**deinit {**

**}**

**@IBAction func skipClick(\_ sender: UIButton) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_skip", extras: EventLog(message: "skip"))**

**self.dismiss(animated: true, completion: nil)**

**NotificationCenter.default.post(Notification(name: Notification.Name("changeRoot")))**

**}**

**func setTextView() {**

**let str1 = "By purchasing a subscription, you agree to the "**

**let str2 = "Terms of Service"**

**let str3 = " and acknowledge the "**

**let str4 = "Privacy Policy"**

**let str5 = """**

**.Your subscription will automatically renew and your Apple ID account will be replaced with the corresponding fee until you cancel it at least 24 hours before the end of the current current subscription period. Cancel your subscription at any time by following these instructions**

**"""**

**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 = 0.5**

**let att = NSMutableAttributedString(string: str as String, attributes: [NSAttributedString.Key.font:UIFont.pingFangSC\_Regular(size: 10),NSAttributedString.Key.foregroundColor:UIColor.colorWithHexColorString("000000", alpha: 0.7),NSAttributedString.Key.paragraphStyle:parStyle])**

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

**let valueStr2 = "PrivacyPolicy://\(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.colorWithHexColorString("2EC849")]**

**textView.attributedText = att**

**}**

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

**if URL.scheme == "TermsofService" {**

**self.clickTerms()**

**return false**

**}else if URL.scheme == "PrivacyPolicy"{**

**self.clickPrivacy()**

**return false**

**}**

**return true**

**}**

**func clickPrivacy() {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_privacy", extras: EventLog(message: "priviacy"))**

**let vc = MN\_PureWebViewController()**

**vc.urlStr = "https://www.callsnotice.com/call/privacy.html"**

**vc.titleStr = "Privacy Policy"**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}**

**func clickTerms() {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_terms", extras: EventLog(message: "terms"))**

**let vc = MN\_PureWebViewController()**

**vc.urlStr = "https://www.callsnotice.com/call/terms.html"**

**vc.titleStr = "Terms of Use"**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}**

**}**

**import UIKit**

**import WebKit**

**import SnapKit**

**class MN\_PureWebViewController: UIViewController,WKNavigationDelegate {**

**var webView:WKWebView?**

**var urlStr:String = ""**

**var titleStr:String = ""**

**lazy var progressView:UIProgressView = {**

**let progressView = UIProgressView()**

**progressView.trackTintColor = UIColor.white**

**progressView.progressTintColor = UIColor(red: 66/255.0, green: 192/255.0, blue: 46/255.0, alpha: 1)**

**return progressView**

**}()**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**initWebToolbar()**

**initWebView()**

**loadWebView()**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**func initWebToolbar(){**

**let toolBar = UIView(frame: CGRect(x: 0, y: 0, width: LJScreenWidth, height: MN\_Navi\_Height))**

**toolBar.backgroundColor = UIColor.colorWithHexColorString("0xFFFFFF")**

**let lineView = UIView(frame: CGRect(x: 0, y: MN\_Navi\_Height - 1, width: LJScreenWidth, height: 1))**

**let goBackButton = UIButton(type: .custom)**

**goBackButton.setImage(UIImage(named: "navback"), for: .normal)**

**goBackButton.addTarget(self, action: #selector(backClicked), for: .touchUpInside)**

**let titleLab = UILabel()**

**if self.titleStr != "" {**

**titleLab.text = self.titleStr**

**}**

**titleLab.textColor = UIColor.black**

**titleLab.font = UIFont.boldSystemFont(ofSize: 17)**

**toolBar.addSubview(goBackButton)**

**toolBar.addSubview(titleLab)**

**toolBar.addSubview(lineView)**

**self.view.addSubview(toolBar)**

**goBackButton.snp.makeConstraints { make in**

**make.left.equalTo(toolBar.snp\_leftMargin).offset(6)**

**make.bottom.equalTo(toolBar.snp\_bottomMargin).offset(-4)**

**make.width.height.equalTo(30)**

**}**

**titleLab.snp.makeConstraints { make in**

**make.centerX.equalTo(toolBar.snp\_centerXWithinMargins)**

**make.bottom.equalTo(toolBar.snp\_bottomMargin).offset(-7)**

**make.height.equalTo(22)**

**}**

**}**

**func initWebView() {**

**self.progressView.frame = CGRect(x: 0, y: MN\_Navi\_Height, width: LJScreenWidth, height: 2)**

**self.view.addSubview(progressView)**

**self.view.backgroundColor = UIColor.white**

**self.webView = WKWebView(frame: CGRect(x: 16, y: MN\_Navi\_Height + 12, width: LJScreenWidth - 32, height: LJScreenHeight - MN\_Navi\_Height - 12))**

**self.webView?.backgroundColor = UIColor.clear**

**self.webView?.navigationDelegate = self**

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

**self.view.addSubview(self.webView!);**

**}**

**func loadWebView() {**

**if self.urlStr != "" {**

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

**self.webView?.load(request)**

**}**

**}**

**@objc func backClicked(){**

**self.dismiss(animated: true, completion: nil)**

**}**

**func webView(\_ webView: WKWebView, decidePolicyFor navigationAction: WKNavigationAction, decisionHandler: @escaping (WKNavigationActionPolicy) -> Void) {**

**progressView.setProgress(0.0, animated: false)**

**if navigationAction.targetFrame == nil {**

**webView .load(navigationAction.request)**

**}**

**decisionHandler(WKNavigationActionPolicy.allow)**

**}**

**func webView(\_ webView: WKWebView, didFinish navigation: WKNavigation!) {**

**progressView.setProgress(0.0, animated: false)**

**}**

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

**if (keyPath == "estimatedProgress") {**

**progressView.isHidden = self.webView?.estimatedProgress == 1**

**progressView.setProgress(Float(self.webView!.estimatedProgress), animated: true)**

**}**

**}**

**}**

**import UIKit**

**import Contacts**

**import CYLTableViewPlaceHolder**

**class MN\_ContactViewController: UIViewController,CYLTableViewPlaceHolderDelegate,EmptyPlaceHoderDelegate,UITableViewDelegate,UITableViewDataSource,UISearchBarDelegate {**

**func buttonClick() {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_setting", extras: EventLog(message: "go\_setting"))**

**let url = URL(string: UIApplication.openSettingsURLString)**

**if let url = url, UIApplication.shared.canOpenURL(url) {**

**UIApplication.shared.open(url, options: [:],**

**completionHandler: {**

**(success) in**

**})**

**}**

**}**

**func makePlaceHolderView() -> UIView! {**

**let emptyTap = UITapGestureRecognizer(target: self, action: #selector(emptyViewTap))**

**self.emptyView.addGestureRecognizer(emptyTap)**

**return self.emptyView**

**}**

**@objc func emptyViewTap(){**

**}**

**lazy var emptyView: MN\_EmptyPlaceHoderView = {**

**let epv = MN\_EmptyPlaceHoderView.loadFromNib("MN\_EmptyPlaceHoderView")**

**epv.delegate = self**

**return epv**

**}()**

**@IBOutlet weak var headTop: NSLayoutConstraint!**

**@IBOutlet var headView: UIView!**

**@IBOutlet weak var tableView: UITableView!**

**var titleView:UILabel?**

**var contactDic:Dictionary<String,[CNContact]> = [:]**

**var allKeys:Array<String> = []**

**var searchRes:Array<CNContact> = []**

**var isSearch = false**

**var currentNum = ""**

**@IBOutlet weak var searchBar: UISearchBar!**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**let titleView = UILabel()**

**titleView.font = UIFont.pingFangSC\_Semibold(size: 16)**

**titleView.text = ""**

**titleView.textColor = .black**

**titleView.textAlignment = .center**

**titleView.lineBreakMode = .byTruncatingMiddle**

**titleView.frame = CGRect(x: 0, y: 0, width: 100, height: 44)**

**self.navigationItem.titleView = titleView**

**self.titleView = titleView**

**MN\_ContactManager.shared.GetAthorization {\_ in**

**}**

**currentCallType = .centerNumber**

**self.tableView.separatorColor = UIColor.colorWithHexColorString("000000", alpha: 0.1)**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**MN\_ContactManager.shared.Contacts\_sort { (contactDic, keys) in**

**self.contactDic = contactDic**

**self.allKeys = keys**

**self.tableView.cyl\_reloadData()**

**}**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**func scrollViewDidScroll(\_ scrollView: UIScrollView) {**

**if scrollView.contentOffset.y < 45 {**

**headTop.constant = -scrollView.contentOffset.y**

**self.titleView?.text = ""**

**}else{**

**headTop.constant = -45**

**self.titleView?.text = "Contacts"**

**}**

**}**

**func numberOfSections(in tableView: UITableView) -> Int {**

**if isSearch {**

**return 1**

**}else{**

**return allKeys.count**

**}**

**}**

**func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {**

**if isSearch {**

**return searchRes.count**

**}else{**

**return contactDic[allKeys[section]]?.count ?? 0**

**}**

**}**

**func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {**

**let cell = tableView.dequeueReusableCell(withIdentifier: "contactCell", for: indexPath)**

**let contact:CNContact**

**if isSearch {**

**contact = searchRes[indexPath.row]**

**}else{**

**contact = contactDic[allKeys[indexPath.section]]?[indexPath.row] ?? CNContact()**

**}**

**cell.textLabel?.text = contact.fullName()**

**cell.textLabel?.font = UIFont.pingFangSC\_Regular(size: 17)**

**cell.textLabel?.textColor = UIColor.black**

**return cell**

**}**

**func tableView(\_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {**

**if isSearch {**

**return UIView()**

**}else{**

**let view = UIView(frame: CGRect(x: 0, y: 0, width: LJScreenWidth, height: 28))**

**view.backgroundColor = UIColor.colorWithHexColorString("#F8F8F8", alpha: 0.82)**

**let lab = UILabel(frame: CGRect(x: 16, y: 0, width: 100, height: 28))**

**lab.text = allKeys[section]**

**lab.font = UIFont.pingFangSC\_Semibold(size: 17)**

**lab.textColor = .black**

**view.addSubview(lab)**

**return view**

**}**

**}**

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

**if isSearch {**

**return 0**

**}else{**

**return 28**

**}**

**}**

**func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {**

**return 44**

**}**

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

**let contact:CNContact**

**if isSearch {**

**contact = searchRes[indexPath.row]**

**}else{**

**contact = contactDic[allKeys[indexPath.section]]?[indexPath.row] ?? CNContact()**

**}**

**guard let phoneNum = contact.phoneNumbers.first?.value.stringValue else{**

**SVProgressHUD.showInfo(string: "phone number is not exist")**

**return**

**}**

**if !MN\_PurchaseTool.sharedInstance.getCurVipStatus() {**

**DispatchQueue.main.async {**

**let vc = MN\_PurchaseViewController.loadStoryboard(name: "Main")**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}**

**return**

**}**

**if !UserDefaults.standard.bool(forKey: "islogin") {**

**let vc = MN\_RegisterViewController.loadStoryboard(name: "Main")**

**vc.block = {**

**self.dismiss(animated: false, completion: nil)**

**}**

**let nav = BaseNavigationController(rootViewController: vc)**

**nav.modalPresentationStyle = .fullScreen**

**self.present(nav, animated: true, completion: nil)**

**return**

**}**

**currentNum = phoneNum.replacingOccurrencesForPhoneNumber()**

**userCallNum = currentNum**

**var callNum = ""**

**if currentCallType == .userNumber {**

**callNum = currentNum**

**}else{**

**callNum = centerNum**

**}**

**let phone = "telprompt://" + callNum**

**if UIApplication.shared.canOpenURL(URL(string: phone)!) {**

**UIApplication.shared.open(URL(string: phone)!, options: [:],**

**completionHandler: {**

**(success) in**

**})**

**}**

**}**

**func sectionIndexTitles(for tableView: UITableView) -> [String]? {**

**if isSearch {**

**return []**

**}else{**

**return allKeys**

**}**

**}**

**func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {**

**if isSearch {**

**return ""**

**}else{**

**return allKeys[section]**

**}**

**}**

**func scrollViewDidChangeAdjustedContentInset(\_ scrollView: UIScrollView) {**

**self.tableView.cyl\_reloadData()**

**self.searchBar.resignFirstResponder()**

**}**

**func tableView(\_ tableView: UITableView, sectionForSectionIndexTitle title: String, at index: Int) -> Int {**

**return index**

**}**

**func searchBar(\_ searchBar: UISearchBar, textDidChange searchText: String) {**

**if searchText.count == 0 {**

**isSearch = false**

**searchBar.resignFirstResponder()**

**tableView.cyl\_reloadData()**

**return**

**}**

**isSearch = true**

**searchRes = []**

**MN\_ContactManager.shared.Contacts(allInfo: true) { contacts in**

**contacts.forEach { contact in**

**if contact.fullName().contains(searchText){**

**self.searchRes.append(contact)**

**}**

**}**

**self.tableView.reloadData()**

**}**

**}**

**func searchBarSearchButtonClicked(\_ searchBar: UISearchBar) {**

**searchBar.resignFirstResponder()**

**}**

**func searchBarResultsListButtonClicked(\_ searchBar: UISearchBar) {**

**searchBar.resignFirstResponder()**

**}**

**}**

**import UIKit**

**import CYLTableViewPlaceHolder**

**import SwiftyJSON**

**import MJRefresh**

**class MN\_RecordViewController: UIViewController,UITableViewDelegate,UITableViewDataSource,StoryboardLoadable,CYLTableViewPlaceHolderDelegate,EmptyPlaceHoderDelegate,RecordCellDelegate {**

**@IBOutlet weak var tableView: UITableView!**

**var currentIndex:Int = -1**

**var recordArray:Array<RecordModel> = []**

**var currentTFFrame:CGRect?**

**var isChange = false**

**func buttonClick() {**

**}**

**func makePlaceHolderView() -> UIView! {**

**let emptyTap = UITapGestureRecognizer(target: self, action: #selector(emptyViewTap))**

**self.emptyView.addGestureRecognizer(emptyTap)**

**return self.emptyView**

**}**

**@objc func emptyViewTap(){**

**}**

**lazy var emptyView: MN\_EmptyPlaceHoderView = {**

**let epv = MN\_EmptyPlaceHoderView.loadFromNib("MN\_EmptyPlaceHoderView")**

**epv.titleLab.text = "Nothing Here"**

**epv.subLab.text = "Your call recordings will be displayed here"**

**epv.headImage.image = UIImage(named: "img\_record\_null")**

**epv.settingBtn.isHidden = true**

**return epv**

**}()**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**FMDBManger.shareManger().openDB(DBName: "Record")**

**self.tableView.frame.size.width = LJScreenWidth**

**self.tableView.mj\_header = MJRefreshNormalHeader(refreshingBlock: {**

**PTEventRecord.shareManager().addEvent(withType: "pulldown", name: "requestrecord", extras: EventLog(message: "reload\_request"))**

**self.requestRecordData()**

**})**

**NotificationCenter.default.addObserver(self,**

**selector: #selector(keyBoardWillShow(notification:)), name: UIResponder.keyboardWillShowNotification,**

**object: nil)**

**NotificationCenter.default.addObserver(self,**

**selector: #selector(keyBoardWillHide(notification:)), name: UIResponder.keyboardWillHideNotification,**

**object: nil)**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**super.viewWillAppear(animated)**

**requestRecordData()**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**}**

**@objc func keyBoardWillShow(notification: Notification) {**

**let userInfo = notification.userInfo! as Dictionary**

**let value = userInfo[UIResponder.keyboardFrameEndUserInfoKey] as! NSValue**

**let keyBoardRect = value.cgRectValue**

**let keyBoardHeight = keyBoardRect.size.height**

**let duration = userInfo[UIResponder.keyboardAnimationDurationUserInfoKey] as! NSNumber**

**isChange = false**

**UIView.animate(withDuration: TimeInterval(truncating: duration), delay: 0, options: [UIView.AnimationOptions(rawValue: 7)], animations: {**

**if ((self.currentTFFrame?.origin.y ?? 0) + 50) > keyBoardRect.origin.y{**

**self.view.frame = CGRect(x: 0, y: -(keyBoardHeight - MN\_Tabbar\_Height - MN\_Navi\_Height), width: LJScreenWidth, height: self.view.frame.height)**

**self.isChange = true**

**}else{**

**self.isChange = false**

**}**

**}, completion: nil)**

**}**

**@objc func keyBoardWillHide(notification: Notification) {**

**let userInfo = notification.userInfo! as Dictionary**

**let duration = userInfo[UIResponder.keyboardAnimationDurationUserInfoKey] as! NSNumber**

**UIView.animate(withDuration: TimeInterval(truncating: duration), delay: 0, options: [UIView.AnimationOptions(rawValue: 7)], animations: {**

**if self.isChange{**

**self.view.frame = CGRect(x: 0, y: MN\_Navi\_Height, width: LJScreenWidth, height: LJScreenHeight - MN\_Tabbar\_Height - MN\_Navi\_Height)**

**}**

**}, completion: nil)**

**}**

**deinit {**

**NotificationCenter.default.removeObserver(self)**

**}**

**func requestRecordData() {**

**SVProgressHUD.showStatus(string: "Loading")**

**self.recordArray = []**

**currentIndex = -1**

**let uuid = FCUUID.uuidForDevice() ?? ""**

**MN\_NetworkingTool.requestWithGET(requestUrl: RecordList,param: ["uuid":uuid], success: { response in**

**if response.code == 403 {**

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

**let jsonArr = JSON(parseJSON: response.jsonStr).arrayValue**

**for json in jsonArr {**

**let model = RecordModel(json: json)**

**FMDBManger.shareManger().insertRecord(record: model)**

**}**

**self.recordArray = []**

**FMDBManger.shareManger().selectAllRecord().forEach { (model) in**

**self.recordArray.append(model)**

**}**

**}**

**self.recordArray.sort(by: {$0.start\_time > $1.start\_time})**

**self.tableView.mj\_header?.endRefreshing()**

**self.tableView.cyl\_reloadData()**

**SVProgressHUD.dismiss()**

**}) { error in**

**self.tableView.mj\_header?.endRefreshing()**

**SVProgressHUD.dismiss()**

**}**

**}**

**func numberOfSections(in tableView: UITableView) -> Int {**

**return 1**

**}**

**func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {**

**return self.recordArray.count**

**}**

**func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {**

**let cell = tableView.dequeueReusableCell(withIdentifier: "recordCell", for: indexPath) as? MN\_RecordTableViewCell**

**if indexPath.row > self.recordArray.count - 1 {**

**return UITableViewCell()**

**}**

**cell?.model = self.recordArray[indexPath.row]**

**cell?.delegate = self**

**return cell ?? UITableViewCell()**

**}**

**func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {**

**if indexPath.row == currentIndex {**

**return 180**

**}else{**

**return 60**

**}**

**}**

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

**self.tableView.beginUpdates()**

**if currentIndex == indexPath.row {**

**currentIndex = -1**

**}else{**

**currentIndex = indexPath.row**

**}**

**self.tableView.endUpdates()**

**}**

**func deleteWith(model: RecordModel) {**

**self.showAlert(title: "Remove the tape") {**

**MN\_NetworkingTool.requestWithPOST(requestUrl: DeleteRecords, param: ["uuid":FCUUID.uuidForDevice() ?? "","records":[model.recording\_uuid]], success: { response in**

**self.currentIndex = -1**

**if model.local\_url != ""{**

**AudioFileManger.deleteFile(fileName: model.local\_url)**

**}**

**FMDBManger.shareManger().deleteRecord(recordId: model.record\_id)**

**self.recordArray = []**

**FMDBManger.shareManger().selectAllRecord().forEach { (model) in**

**self.recordArray.append(model)**

**}**

**self.tableView.cyl\_reloadData()**

**}) { error in**

**}**

**}**

**}**

**func showMoreWith(model: RecordModel) {**

**let filePathDOC = AudioFileManger.filePath + "/\(model.local\_url)"**

**let url = URL(fileURLWithPath: filePathDOC)**

**let activityController = UIActivityViewController(activityItems: [url], applicationActivities: [])**

**self.present(activityController, animated: true) { }**

**}**

**func titleBeginEdit(fieldFrame: CGRect) {**

**self.currentTFFrame = fieldFrame**

**}**

**}**

**import UIKit**

**import CallKit**

**import PhoneNumberKit**

**enum CallNumType {**

**case userNumber**

**case centerNumber**

**}**

**class MN\_DialViewController: UIViewController {**

**@IBOutlet weak var numberLab: UILabel!**

**@IBOutlet weak var deleteBtn: UIButton!**

**var phoneNumberKit = PhoneNumberKit()**

**var callNum:String = ""{**

**didSet{**

**guard let phoneNumber = try? phoneNumberKit.parse(callNum) else {**

**numberLab.text = callNum**

**return**

**}**

**numberLab.text = phoneNumberKit.format(phoneNumber, toType: .international)**

**}**

**}**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**numberLab.adjustsFontSizeToFitWidth = true**

**currentCallType = .centerNumber**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**}**

**@IBAction func zeroLongPress(\_ sender: UILongPressGestureRecognizer) {**

**PTEventRecord.shareManager().addEvent(withType: "longpress", name: "btn\_zero", extras: EventLog(message: "longpress"))**

**if sender.state == .began{**

**callNum = callNum + "+"**

**deleteBtn.isHidden = false**

**}**

**}**

**@IBAction func numberClick(\_ sender: UIButton) {**

**var str = ""**

**if sender.tag == 101 {**

**str = "\*"**

**}else if sender.tag == 102{**

**str = "#"**

**}else{**

**str = "\(sender.tag)"**

**}**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_dial", extras: EventLog(message: "str"))**

**callNum = callNum + str**

**if callNum != "" {**

**deleteBtn.isHidden = false**

**}**

**}**

**@IBAction func callBtnClick(\_ sender: UIButton) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_call", extras: EventLog(message: "callClick"))**

**if !MN\_PurchaseTool.sharedInstance.getCurVipStatus() {**

**let vc = MN\_PurchaseViewController.loadStoryboard(name: "Main")**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**return**

**}**

**if !UserDefaults.standard.bool(forKey: "islogin") {**

**let vc = MN\_RegisterViewController.loadStoryboard(name: "Main")**

**vc.block = {**

**self.dismiss(animated: false, completion: nil)**

**}**

**let nav = BaseNavigationController(rootViewController: vc)**

**nav.modalPresentationStyle = .fullScreen**

**self.present(nav, animated: true, completion: nil)**

**return**

**}**

**if numberLab.text == "" {**

**SVProgressHUD.showInfo(string: "plase input number")**

**return**

**}**

**var callNum = ""**

**if currentCallType == .userNumber {**

**callNum = self.callNum**

**}else{**

**callNum = centerNum**

**}**

**userCallNum = self.callNum**

**let phone = "tel://" + callNum**

**if UIApplication.shared.canOpenURL(URL(string: phone)!) {**

**UIApplication.shared.open(URL(string: phone)!, options: [:],**

**completionHandler: {**

**(success) in**

**})**

**}**

**}**

**@IBAction func deleteBtnClick(\_ sender: UIButton) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_delete", extras: EventLog(message: "deleteClick"))**

**if self.callNum.count > 0 {**

**self.callNum.removeLast()**

**}**

**if self.callNum.count == 0 {**

**self.deleteBtn.isHidden = true**

**}**

**}**

**}**

**import UIKit**

**class MN\_MeTableViewController: UITableViewController {**

**@IBOutlet weak var versionLab: UILabel!**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**self.tableView.separatorColor = UIColor.colorWithHexColorString("000000", alpha: 0.1)**

**versionLab.text = "\(Bundle.main.infoDictionary!["CFBundleShortVersionString"] as! String)"**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "leave", name: "page", extras: EventLog(message: ""))**

**}**

**override func viewWillAppear(\_ animated: Bool) {**

**PTEventRecord.shareManager().addEvent(withType: "enter", name: "page", extras: EventLog(message: ""))**

**}**

**@IBAction func outCallClick(\_ sender: UITapGestureRecognizer) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_outCall", extras: EventLog(message: "outcall"))**

**let vc = MN\_MeGuideViewController.loadStoryboard(name: "Main")**

**vc.navTitle = "Outgoing Calling Guide"**

**self.navigationController?.pushViewController(vc, animated: true)**

**}**

**@IBAction func inCallClick(\_ sender: UITapGestureRecognizer) {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_inCall", extras: EventLog(message: "incall"))**

**let vc = MN\_MeGuideViewController.loadStoryboard(name: "Main")**

**vc.navTitle = "Incoming Calling Guide"**

**vc.scrollView.images = ["img\_incoming\_setp1","img\_incoming\_setp2","img\_incoming\_setp3"]**

**vc.scrollView.titles = ["Answer phone","Call recording center first","Merge calls"]**

**vc.scrollView.subTitles = ["You will be asked whether record imcoming \ncall or not while phone is ringing","Dial the recording center number in the \npop-up menu","Touch Merge when connected to recording \ncenter and speak freely, everything you and \nyour friend saying will be recorded"]**

**self.navigationController?.pushViewController(vc, animated: true)**

**}**

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

**if indexPath.row == 1 {**

**let vc = MN\_RegisterViewController.loadStoryboard(name: "Main")**

**vc.block = {**

**self.dismiss(animated: false, completion: nil)**

**}**

**let nav = BaseNavigationController(rootViewController: vc)**

**nav.modalPresentationStyle = .fullScreen**

**self.present(nav, animated: true, completion: nil)**

**}else if indexPath.row == 2 {**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "cell\_about", extras: EventLog(message: "see about"))**

**let vc = MN\_PureWebViewController()**

**vc.urlStr = "https://www.callsnotice.com/call/index.html"**

**vc.titleStr = "About Us";**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}else if indexPath.row == 3{**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "cell\_share", extras: EventLog(message: "shareApp"))**

**let image = UIImage(named: "share\_icon.png")**

**let url = URL(string: "https://apps.apple.com/cn/app/id1533760922")!**

**let activityController = UIActivityViewController(activityItems: ["Call Recorder - for iPhone", image!,url], applicationActivities: [])**

**activityController.completionWithItemsHandler = {**

**(type, flag, array, error) -> Swift.Void in**

**}**

**present(activityController, animated: true) { }**

**}else if indexPath.row == 4{**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "cell\_rating", extras: EventLog(message: "rating"))**

**let appUrl = "https://itunes.apple.com/cn/app/id1533760922?action=write-review"**

**if UIApplication.shared.canOpenURL(URL(string: appUrl)!){**

**UIApplication.shared.open(URL(string: appUrl)!, options: [:], completionHandler: nil)**

**}**

**}else if indexPath.row == 5{**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "cell\_restore", extras: EventLog(message: "restore"))**

**MN\_PurchaseTool.sharedInstance.restorePurchases { res in**

**if res{**

**if !UserDefaults.standard.bool(forKey: "islogin") {**

**let vc = MN\_RegisterViewController.loadStoryboard(name: "Main")**

**vc.block = {**

**self.dismiss(animated: false, completion: nil)**

**}**

**DispatchQueue.main.async {**

**let nav = BaseNavigationController(rootViewController: vc)**

**nav.modalPresentationStyle = .fullScreen**

**self.present(nav, animated: true, completion: nil)**

**}**

**}else{**

**DispatchQueue.main.async {**

**self.dismiss(animated: true, completion: nil)**

**NotificationCenter.default.post(Notification(name: Notification.Name("changeRoot")))**

**}**

**}**

**}**

**}**

**}else if indexPath.row == 7{**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "cell\_privacy", extras: EventLog(message: "priviacy"))**

**let vc = MN\_PureWebViewController()**

**vc.urlStr = "https://www.callsnotice.com/call/privacy.html"**

**vc.titleStr = "Privacy Policy";**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}else if indexPath.row == 8{**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "cell\_terms", extras: EventLog(message: "terms"))**

**let vc = MN\_PureWebViewController()**

**vc.urlStr = "https://www.callsnotice.com/call/terms.html"**

**vc.titleStr = "Terms of Use";**

**vc.modalPresentationStyle = .fullScreen**

**self.present(vc, animated: true, completion: nil)**

**}**

**}**

**}**

**import UIKit**

**class MN\_MeGuideViewController: UIViewController,StoryboardLoadable,UIScrollViewDelegate {**

**@IBOutlet var guideBottom: UIView!**

**lazy var scrollView: MN\_GuideScrollView = {**

**let scroll = MN\_GuideScrollView.loadFromNib()**

**scroll.frame = CGRect(x: 0, y: 0, width: LJScreenWidth, height:  LJScreenHeight - MN\_Navi\_Height - MN\_TabbarSafeBottomMargin - 90)**

**scroll.setRestoreHidden()**

**return scroll**

**}()**

**var mainScrollerView: UIScrollView!**

**var navTitle = ""**

**override func viewWillAppear(\_ animated: Bool) {**

**self.tabBarController?.tabBar.isHidden = true**

**}**

**override func viewWillDisappear(\_ animated: Bool) {**

**self.tabBarController?.tabBar.isHidden = false**

**}**

**override func viewDidLoad() {**

**super.viewDidLoad()**

**let scroll = UIScrollView(frame: CGRect(x: 0, y: 0, width: LJScreenWidth, height: LJScreenHeight))**

**scroll.contentSize = CGSize(width: LJScreenWidth, height: LJScreenHeight)**

**scroll.delegate = self**

**self.view.addSubview(scroll)**

**scroll.addSubview(self.scrollView)**

**mainScrollerView = scroll**

**MN\_makeBottomView()**

**let titleView = UILabel()**

**titleView.font = UIFont.pingFangSC\_Semibold(size: 17)**

**titleView.lineBreakMode = .byTruncatingMiddle**

**titleView.text = navTitle**

**self.navigationItem.titleView = titleView**

**}**

**func MN\_makeBottomView(){**

**guideBottom.frame = CGRect(x: 0, y: self.scrollView.frame.origin.y + self.scrollView.frame.size.height , width: LJScreenWidth, height: 90 + MN\_TabbarSafeBottomMargin )**

**mainScrollerView.addSubview(guideBottom)**

**}**

**@IBAction func nextClick(\_ sender: UIButton) {**

**let ofset = self.scrollView.scrollView.contentOffset.x**

**if ofset < 2 \* LJScreenWidth {**

**self.scrollView.scrollView.scrollRectToVisible(CGRect(x: ofset + LJScreenWidth, y: 0, width: LJScreenWidth, height: self.scrollView.scrollView.contentSize.height), animated: true)**

**}else if ofset == 2 \* LJScreenWidth{**

**self.scrollView.scrollView.scrollRectToVisible(CGRect(x: 0, y: 0, width: LJScreenWidth, height: self.scrollView.scrollView.contentSize.height), animated: true)**

**}**

**PTEventRecord.shareManager().addEvent(withType: "touch", name: "btn\_next", extras: EventLog(message: "nextPage"))**

**}**

**}**