TECHSIGN RKYC

iOS

Change Log

03.07.2023

Pod is updated. New framework face_mesh_detection.xcframework added for liveness head gesture hint implementation.

pod 'techsigncloudlibs', '1.7.7-wrapper', :source => 'https://github.com/Techsign/PODSPECS.git'

id_card_detection_ios_cnn — v1.2.2

- Disable card detection configuration added. When enabled SDK will return the whole camera image not captured card image so not recommended for KYC.

  conf!.disableCardDetection = true // default false

- Flash support is added as configuration. Not recommended for KYC.

  configuration.flashOn = true // You may start the capture with flash on.

  configuration.flashToggleAvailable = true // You may enable the flash toggle button.

- Support for different size unknown card types (blue card, foreign id etc.) added. For example if you set both BLUE_CARD and NEW_BLUE_CARD, SDK will capture both of them.
- NEW_BLUE_CARD card type added.

id_card_detection_ios_wrapper — v1.1.2

- Disable card detection configuration added. When enabled SDK will return the whole camera image not captured card image so not recommended for KYC.

  conf!.disableCardDetection = true // default false

- Flash support is added as configuration. Not recommended for KYC.

  configuration.flashOn = true // You may start the capture with flash on.

  configuration.flashToggleAvailable = true // You may enable the flash toggle button.

- Support for different size unknown card types (blue card, foreign id etc.) added. For example if you set both BLUE_CARD and NEW_BLUE_CARD, SDK will capture both of them.
- NEW_BLUE_CARD card type added.

RKYC_iOS — v1.2.1

- Liveness head gesture hint implementation added for HEAD-HORIZONTAL and HEAD-VERTICAL gestures using face landsmark AI model. A hint appears to guide to user to turn his/her head to required direction and when the user turns the thickness of hint is increased and when the first way(left for HEAD-HORIZONTAL, up for HEAD-VERTICAL) is done the hint appears on next way (right for HEAD-HORIZONTAL, down for HEAD-VERTICAL). If the gesture is completed successfully SDK will return the captured video, if not SDK will return a callback and expects a retry.

// configurations to activate gesture hint

LivenessController.IS_GESTURE_CONTROL_ENABLED = true // default is false

// or .HEAD_VERTICAL, .EYE, .MOUTH

LivenessController.GESTURE_TYPE = .HEAD_HORIZONTAL

// new events for FragmentListener

func positionSucceed(model:PositionSuccedModel) // called when a position (left, up etc.) successfully completed, you may guide user for next position

func positionFailed(model:PositionFailedModel) // called when a position (left, up etc.) failed, you may guide user to turn his/her head to required direction

func positionCorrect() // called when user's head is in correct direction, may be used to clear the guide message

func turnedTooMuch(model:TurnedTooMuchModel) // called when the head degree is too much for detection, you may guide the user to make a slight head movement

func gestureFailed() // called after the recording completed but the required gesture is not completed, you may show the guidance again and restart the process

// UI customizations

LivenessController.GESTURE_HINT_WIDTH = 40 // base width of gesture hint default value is 40

LivenessController.FACE_GESTURE_HINT_COLOR = .cyan // default value is UIColor.orange.cgColor

passport_reader — v1.1.7

- Passport crash on do99 step for some passport fixed.

05.04.2023

Pod is updated

pod 'techsigncloudlibs', '1.6.9-wrapper', :source => 'https://github.com/Techsign/PODSPECS.git'

id_card_detection_ios_cnn — v1.1.8

- Capture problem for capture only in hint configuration fixed and margin configuration added for it.

  conf!.captureOnlyInHint = true // default false

  conf!.captureOnlyInHintMargin = 0.1 // default 0

- Disable four point transform configuration added.

  conf!.disableFourPointTransformation = true // default false

- Capture problem for PDF_A4 fixed.
- Blur detection AI model enhanced.

id_card_detection_ios_wrapper — v1.0.9

- Capture problem for capture only in hint configuration fixed and margin configuration added for it.

  conf!.captureOnlyInHint = true // default false

  conf!.captureOnlyInHintMargin = 0.1 // default 0

- Disable four point transform configuration added.

  conf!.disableFourPointTransformation = true // default false

RKYC_iOS — v1.1.8

- Hologram and liveness memory leak fixed.

passport_reader — v1.1.6

- Passport reader requires 9 character serial number problem fixed for the passports with 8 character serial number ie. Ukrainian and Swedish passports.

29.11.2022

Pod is updated

pod 'techsigncloudlibs', '1.5.3-wrapper', :source => 'https://github.com/Techsign/PODSPECS.git'

id_card_detection_ios_cnn — v1.1.2

- Blur detection AI model enhanced.

id_card_detection_ios_wrapper — v1.0.4

- Max output image size configuration added. If the captured image size exceeds the limit (if any), the image is compressed just below the limitation (if any).

```
// default value is nil so, there is no compression by default

configuration.maxOutputImageSizeInBytes = 1900000
```

RKYC_iOS — v1.1.4

- Max output video size configuration added. If the configuration set the video is compressed just below the limitation (if any).

```
// default value is nil so, there is no compression by default

HologramController.MAX_OUTPUT_FILE_SIZE_IN_BYTES = 1900000

// default value is nil so, there is no compression by default

LivenessController.MAX_OUTPUT_FILE_SIZE_IN_BYTES = 1900000
```

- Miscalculation in hologram margin ratio leading to easy cancellation of the video recording bug fixed.
- Embedded import UIKit removed, you may need to import it.

passport_reader — v1.1.5

- NFC_TRANSMIT_SLEEP_TIME_IN_MICROSECONDS configuration added. You may set a lower timeout to speed up the NFC process but lower timeouts may lead to more connection errors. Minimum recommended timeout is 50000 microseconds.

```
// default value is 100000 since the V3

LocalNFCCommandTransmitter.NFC_TRANSMIT_SLEEP_TIME_IN_MICROSECONDS = 100000
```

14.10.2022

Pod is updated

pod 'techsigncloudlibs', '1.2.6-wrapper', :source => 'https://github.com/Techsign/PODSPECS.git'

id_card_detection_ios_cnn — v1.1.0

- New blur detection algorithm implemented.
- iOS 16 landscape orientation bug fixed.

Id_card_detection_ios_wrapper — v1.0.4

- Optional double focus score threshold configuration added. When isFocusScoreThresholdActive is set to true and the focus score is higher than its threshold(*focusScoreThreshold*) the image is captured immediately. If no frame score is over *focusScoreThreshold* and focus score is over *minFocusScore* more than *eligibleFocusScoreCount* the best focus scored frame is captured. We recommend below configurations.

```
// capture any frame with more than 0.95 focus score

// capture best focus scored frame when consecutive 10 frame focus score is between 0.8 - 0.95

conf.isFocusScoreThresholdActive = true // enable double threshold, default false

conf.focusScoreThreshold = 0.95 // set first threshold to immediately capture

conf.minFocusScore = 0.8 // set second threshold

conf.eligibleFocusScoreCount = 10 // set second threshold count
```

RKYC_iOS — v1.1.4

- HologramController face detection hint margin ratio is set to configurable.

  HologramController.MARGIN_RATIO = 0.2

passport-reader — v1.1.3

- TagError.USER_CANCELED called more than one bug fixed.

19.09.2022

Pod is updated

pod 'techsigncloudlibs', '1.1.7-wrapper, :source => 'https://github.com/Techsign/PODSPECS.git'

id_card_detection_ios_cnn — v1.0.6

- Capture problem for FOREIGN_ID_PORTRAIT and BLUE_CARD fixed.

RKYC_iOS — v1.1.3

- Hologram does not restart when USE_CARD_DETECTION set to true bug fixed.
- Now you can set ApiKey (a token that has longer valid time) for backend requests.

  Authentication.apiKey = “<API_KEY>”

- Add secondary template id card ServerCall implemented.

passport-reader — v1.1.2

- Restart NFC reading without closing the session (NFC popup) configuration added.

```
// set the configuration

PassportReader.USE_NFC_EVENT_DELEGATE = true

func error(error: Error) {

        // suppress error if it is a ConnectionError or AuthenticationFailed

        if(supressError){

                // do not invalidate session just show a message on screen

                self.localCommandTransmitter?.showAlertMessage(alertMessage:
        NSLocalizedString("please_do_not_move_away_your_id", comment: ""))

        }else{

                // invalidate session if it is another error

                self.localCommandTransmitter?.invalidate(errorMessage: errorMessage)

        }

}
```

02.08.2022

Pod is updated

pod 'techsigncloudlibs', '1.0.8-wrapper, :source => 'https://github.com/Techsign/PODSPECS.git'

id_card_detection_ios_wrapper — v1.0.3

- Blur detection model improved. Default and suggested values of below configurations changed as below.

    configuration.minFocusScore = 0.8
    configuration.eligibleFocusScoreCount = 0

- Image correction model implemented. 180° reverse taken pictures corrected then returned on cardDetected.

RKYC_iOS — v1.1.1

- HologramViewDelegate that returns the rectangle of id hint added to HologramController. You can use it to display a custom hint.

```
hologramController.setViewDelegate(self)

public func onViewDrawn(mainRect: CGRect) {
        // mainRect position of card rect in hologram container
}
```

- LivenessViewDelegate that returns the rectangle of face hint added to LivenessController. You can use it to display a custom hint.

```
livenessController.setViewDelegate(self)

public func onViewDrawn(mainRect: CGRect) {
        // mainRect position of card rect in hologram container
}
```

28.06.2022

Pod is updated

pod 'techsigncloudlibs', '1.0.4-wrapper, :source => 'https://github.com/Techsign/PODSPECS.git'

id_card_detection_ios_wrapper — v1.0.2

- captureOnlyInHint configuration added to IDCardDetectionWrapperConfiguration.

  configuration.captureOnlyInHint = true

  - You can also receive the isInsideOfHint info from cardDetectionFailed

  ```
  public func cardDetectionFailed(wrapperErrorsModel: WrapperErrorsModel){

        wrapperErrorsModel.isInsideOfHint // is inside of hint

  }
  ```

- Trademark logo now can be turned off using the below configuration.

  configuration.isTrademarkLogoOn = false

- eligibleFocusScoreCount configuration added to IDCardDetectionWrapperConfiguration.

  ```
  // number of consecutive frames that exceeds minFocusScore for detection
  // default value is 10

  configuration.eligibleFocusScoreCount = 10
  ```

RKYC_iOS — v1.1.0

- Background color configuration for outside of hint added to Hologram and Liveness.

  ```
  HologramController.OUTSIDE_OF_HINT_COLOR = UIColor.black.withAlphaComponent(0.5)
  LivenessController.OUTSIDE_OF_HINT_COLOR = UIColor.black.withAlphaComponent(0.5)
  ```

- Hologram now can start with card detection for iOS13+

HologramController.USE_CARD_DETECION = true

- Cards without photos also can be captured. Now you can set HologramContoller.CONTROL_FACE_CAPTURE to true for cards without photos.

HologramContoller.CONTROL_FACE_CAPTURE = true

- HologramController.STOP_VIDEO_WHILE_RECORDING should be set to false.

HologramController.STOP_VIDEO_WHILE_RECORDING = false

- Card type detection added, only given card type will be captured.

passport_reader — v1.1.1

- Fix some NFC sessions not closed bug.

Minimum SDK

- Supported minimum SDK version is iOS 10.0 that means all devices starting from iPhone 5 are supported which is approximately 99% of devices according to Apple App Store.

Integration

A) Using Pod

pod 'techsigncloudlibs', '1.0.2-wrapper', :source => 'https://github.com/Techsign/PODSPECS.git'

- Our pod repository is private, please contact us for authorisation.
- If you receive an error like below

“CocoaPods could not find compatible versions for pod "techsigncloudlibs":
In Podfile:
“

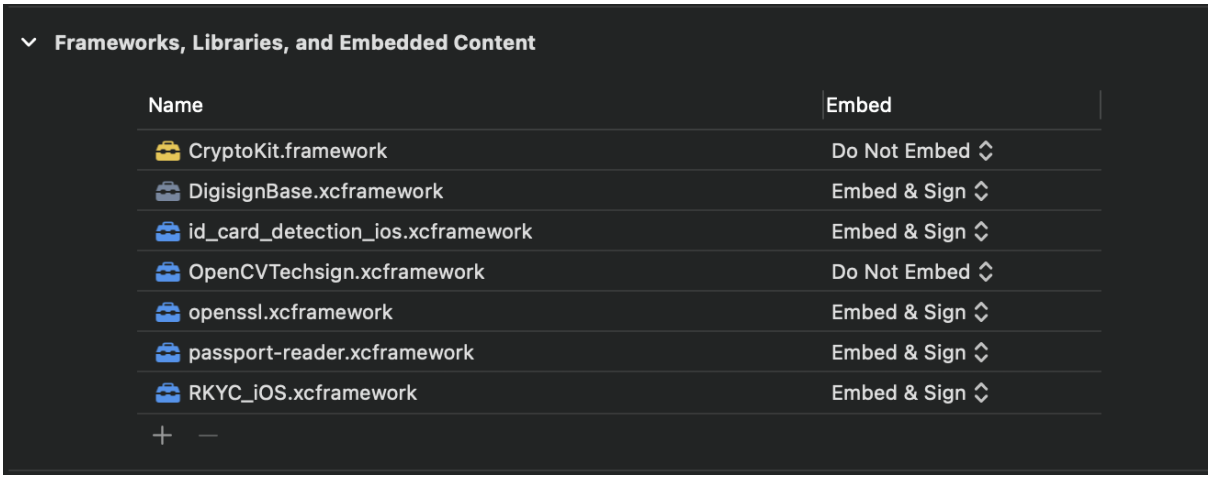
Run following command on terminal

pod update techsigncloudlibs

- The pod repository is only supported by Xcode13+, please contact us if you use an older version of Xcode.

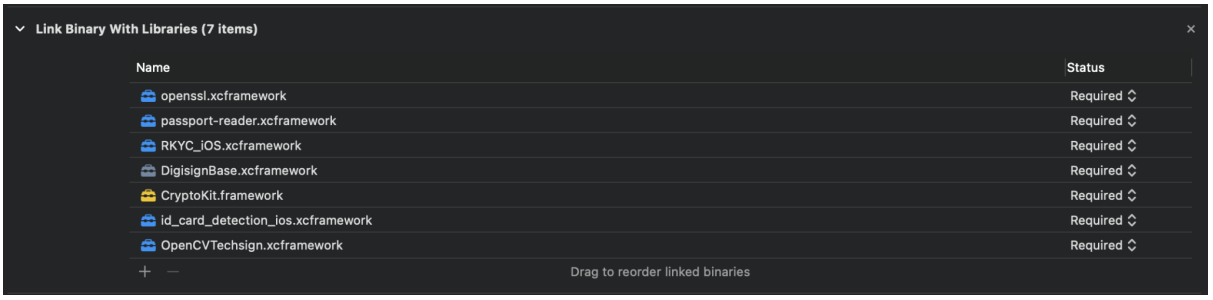
B) Using xcframeworks

There are 8 xcframeworks namely DigisignBase, OpenCVTechsign, id_card_detection_ios, id_card_detection_ios_cnn, id_card_detection_ios_wrapper, openssl_techsign, passport_reader and and RKYC_iOS. These frameworks should be added to the project folder and the folder path should be added to Build Settings -> Framework Search Paths. After that Frameworks, Libraries and Embedded Content in the General tab should look like this;

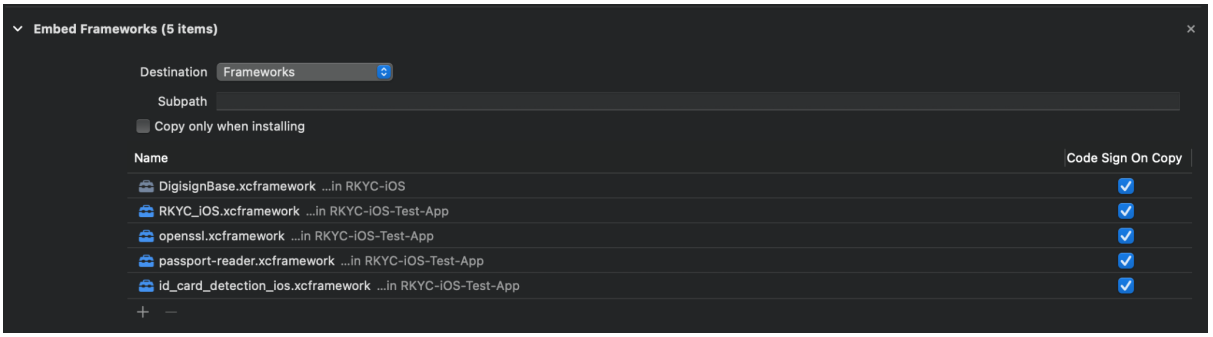


We also add the iOS CoreNFC framework for the use of NFC.

After that, Build Phases -> Link Binary With Libraries should look like this;

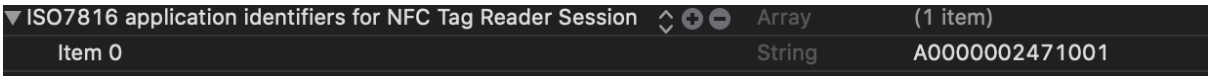


And Build Phases -> Embed Frameworks should look like this;



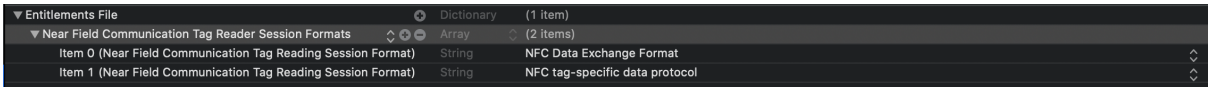
NFC Configurations

1-) The below key value pairs should be added to the info.plist file.



2-) A Code Signing Entitlements file should be added to project => Build Settings

3-) The Code Signing Entitlements file should contain below key value pairs.



API

Test API Root URL:

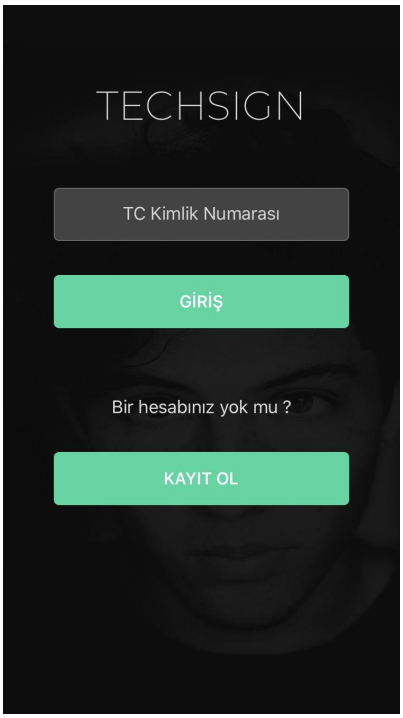
http://169.62.55.86:8098/

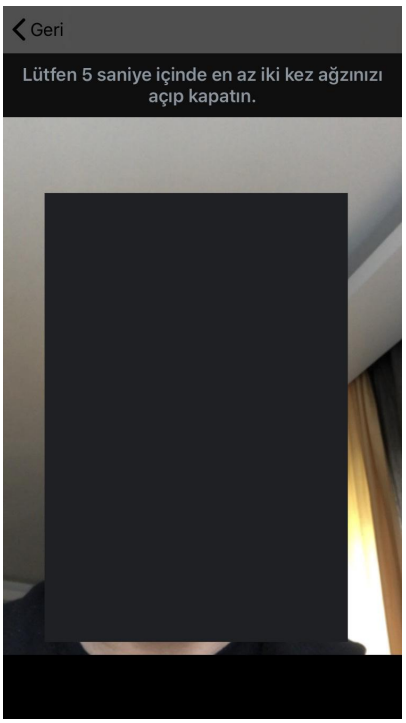
RKYC Server API is based on REST. API calls used for operations are listed in the following sections, together with the corresponding operation. For every API call, the type of the operation (POST/GET), the input and output models are listed, which are typically JSON unless mentioned otherwise.

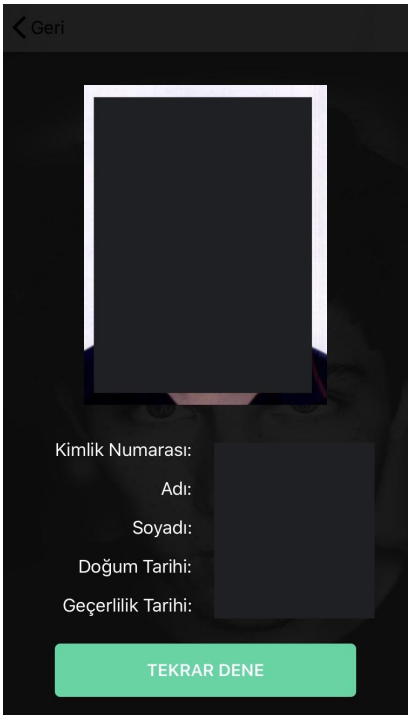
Usage

Our service calls are on ServerCall class, the service calls run async and take a (Result -> Void) function which calls onSuccess method when it is successful and calls onFailure method when there is an error.

1- Login







Login service can be used by calling ServerCall.login() function which requires a LoginModel and a Result<VerifyFaceReturnModel,NSError>.

LoginModel consists of three fields,

- String “id” => citizen id
- String “video” => base64 of recorded selfie video[1]
- int “rotate” => rotation of recorded video (0,90,180 or 270)

[1]. The recorded video should be around 5 seconds and 480p.

If the service call is successful, onSuccess method called with a VerifyFaceReturnModel which consist of,

- String “id” => citizen id
- String “name” => user’s name
- String “surname” => user’s surname
- String “birthDate” => user’s birth date[2]
- String “expireDate” => user’s expire date[2]
- String “picture” => base64 of user’s picture on identity card which he/she used for registration
- String “gesture” => which gesture is used for liveness check[3]
- [ControlEntryModel] “controlResults” => ControlEntryModel consists of,
    - String “entryType” => ControlEntryTypes enum[4]
    - Float “entryScore” => the result of the control between 0 and 1.
    - Boolean “entryInterpretation” => true if the control is successful
    - String “controlId” => unique identifier of the control

[2]. dates are in YYMMDD format.

[3]. we have two gesture types,

- “MOUTH” => user should open and close his/her mouth at least two types in 5 seconds.
- “EYE” => user should open and close his/her eyes at least two types in 5 seconds.

[4].public enum ControlEntryTypes : String{

```
case PhotoCheatVerification = "PHOTOCHEAT"
case HologramVerification = "HOLOGRAM"
case HologramFaceVerification = "HOLOGRAMFACE"
case FaceLivenessVerification = "FACELIVENESS"
case FaceVerification = "FACE"
case HiddenPhotoVerification = "HIDDENPHOTO"
case SignaturePhotoVerification = "SIGNATUREPHOTO"
case GuillocheVerification = "GUILLOCHEVERIFICATION"
case RainbowVerification = "RAINBOWVERIFICATION"
case OCRNFCSimilarity = "OCRNFCSIMILARITY"
case OCRMRZSimilarity = "OCRMRZSIMILARITY"
```

}

We are using the “EYE” gesture in login.

Note: For a successful login both *“*FaceVerification*”* and *“*FaceLivenessVerification*”* in “controlResults” should be passed.

API Call:

Server Suffix: /face/authorize

Type: POST

Input:

{
id:String //ID Number for login, should be registered before
video: String //captured liveness video, in Base64 format
rotate: Int //clock-wise rotation to apply to the video before control, 0/90/180/270
}

Output:

VerifyFaceReturnModel, details listed above.

2- Register

Start Transaction (Optional)

Start transaction service takes an optional data dictionary and makes its job ,if there is any, then returns the *transactionId* and a *gestureType* which is optional and will be used for the liveness process.

You can start the transaction by calling *ServerCall.startTransaction()*

API Call:

Server Suffix: /id/start

Type: POST

Input:

{
optionalData : {}
}

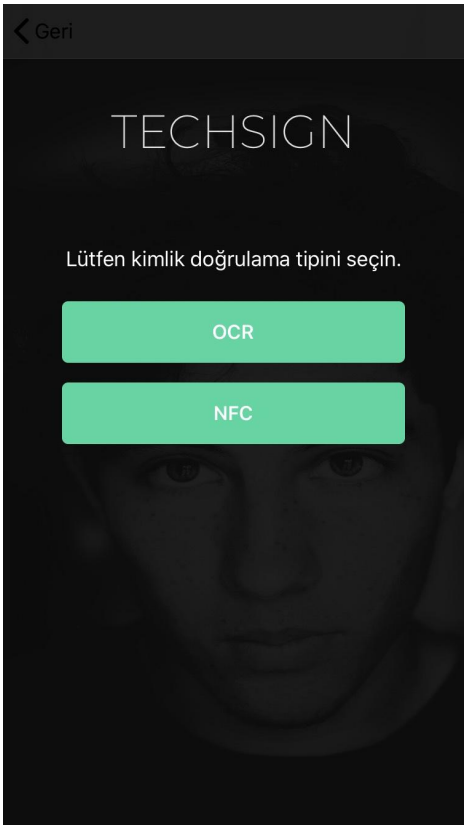
Output:

{
transactionId:String
gestureType:String[1]

}

[1]. we have two gesture types,

- “MOUTH” => user should open and close his/her mouth at least two types in 5 seconds.
- “EYE” => user should open and close his/her eyes at least two types in 5 seconds.



Registration process starts with the card process and ends with the liveness process.

A. Card Process

To register, we capture your card image. You can use *“IDCardDetectionWrapper”* for this.

“IDCardDetectionWrapper” is a wrapper that wraps our old card detection library(uses OpenCV) and new card detection library(uses machine learning models for iOS13+) according to the iOS version.

“IDCardDetectionWrapper” returns a view that fills its container view (parent, superview) so it should be added as a subview to a container view.

```
import id_card_detection_ios_wrapper

// remove import id_card_detection_ios
// remove import id_card_detection_ios_cnn

 class ViewController: UIViewController, IDCardDetectionWrapperDelegate,
IDCardDetectionWrapperViewDelegate {

        @IBOutlet weak var containerView:UIView! // we recommend a container view to embed
our view

    private var wrapper:IDCardDetectionWrapper? // should be an instance otherwise delegates
won't call because of weak reference

    let cardTypes:[WrapperCardType] = [.NEW_ID, .NEW_ID_BACK, .NEW_DRIVER,
.DRIVER_BACK]

      override func viewDidLoad() {
          super.viewDidLoad()

          wrapper = IDCardDetectionWrapper(frame: containerView.frame) // initialize wrapper

          let view = wrapper!.getView() // get view according to iOS version (cnn for iOS13+,
otherwise opencv)

          self.containerView.addSubview(view!)


}
```

Configurations should be set using the "*IDCardDetectionWrapperConfiguration*" class and "*setConfiguration*" function of "IDCardDetectionWrapper".

```
 conf = IDCardDetectionWrapperConfiguration()

// card type list should be given
// default value is  [.NEW_ID, .NEW_ID_BACK, .NEW_DRIVER, .DRIVER_BACK]
// should give same size cards for below ios13, otherwise throws fatal error
conf!.cardTypeList = cardTypes

wrapper!.setConfiguration(conf: conf!)
```

Note: see Customizations -> OCR on the last page of the document for other configurations (text recognizing options and visual customizations).

Set the "*IDCardDetectionWrapperDelegate*"

```
    wrapper!.setDelegate(delegate: self)
```

Implement the "*IDCardDetectionWrapperDelegate".*

```
// called when the card is captured, returns cropped image and card type

public func cardDetected(image: UIImage, cardType: WrapperCardType){
        wrapper?.stopCapture() // stop capture to avoid multiple captures
}

// called every unsuccessful frame
// you may inform user about the detection using wrapperErrorsModel

public func cardDetectionFailed(wrapperErrorsModel: WrapperErrorsModel){


    DispatchQueue.main.async {

        if(wrapperErrorsModel.cardType == WrapperCardType.NO_CARD){
            // no card detected idle case
        }else if(!self.cardTypes.contains(wrapperErrorsModel.cardType)){
            // unaccepted card type found
        }else{
            if(!wrapperErrorsModel.isXPerspectiveEligible){
                if(wrapperErrorsModel.xPerspectiveErrorRatio < 0){
                    // left edge perspective is not eligible
                    // left edge is way shorter than right edge
                }else{
                    // right edge perspective is not eligible
                    // right edge is way shorter than left edge
                }
            }else if(!wrapperErrorsModel.isYPerspectiveEligible){
                if(wrapperErrorsModel.yPerspectiveErrorRatio < 0){
                    // top edge perspective is not eligible
                    // top edge is way shorter than bottom edge
                }else{
                    // bottom edge perspective is not eligible
                    // bottom edge is way shorter than top edge
                }
            }else if(!wrapperErrorsModel.isWidthHeightRatioEligible){
                if(wrapperErrorsModel.widthHeightErrorRatio < 0){
                    // width/height ratio is too small
                    // width is way smaller than expected
                }else{
                    // width/height ratio is too big
                    // height is way smaller than expected
                }
            }else if(!wrapperErrorsModel.isAreaRatioEligible){
                if(wrapperErrorsModel.areaErrorRatio < 1){
                    // card is too far away
                }else{
                    // card is too close
                }
            }else if(!wrapperErrorsModel.isImageSharpnessRatioEligible){
                // image is blurry
```

```
        }
      }
    }

  }
```

Then, start the capturing process.

```
wrapper?.startCapture()
```

Notes:
1-) You can stop the capture when you receive the result on *cardDetected*(), then change the configuration and start the capture again.

```
wrapper?.stopCapture()
conf!.hintColor = UIColor.black.cgColor
wrapper?.setConfiguration(conf: conf!)
wrapper?.startCapture()
```

3-) If you want to show a custom view on the "*IDCardDetectionView*" you need to bring the subview to the front.

```
containerView.bringSubviewToFront(customView)
```

2-) The orientation of the view is adaptive to the orientation of the UIViewController, we suggest the portrait orientation for all card types. You can use these utility functions to lock the orientation.

AppUtility

```
import Foundation
import UIKit
struct AppUtility {

        static func lockOrientation(_ orientation: UIInterfaceOrientationMask) {

                if let delegate = UIApplication.shared.delegate as? AppDelegate {
                        delegate.orientationLock = orientation
                }
        }

/// OPTIONAL Added method to adjust lock and rotate to the desired orientation
static func lockOrientation(_ orientation: UIInterfaceOrientationMask, andRotateTo
rotateOrientation:UIInterfaceOrientation) {

                self.lockOrientation(orientation)
```

```
                UIDevice.current.setValue(rotateOrientation.rawValue, forKey: "orientation")
                UINavigationController.attemptRotationToDeviceOrientation()
        }

}
```

AppDelegate

```
var orientationLock = UIInterfaceOrientationMask.all // app's default

        func application(_ application: UIApplication, supportedInterfaceOrientationsFor
window: UIWindow?) -> UIInterfaceOrientationMask {
                return self.orientationLock
        }
```

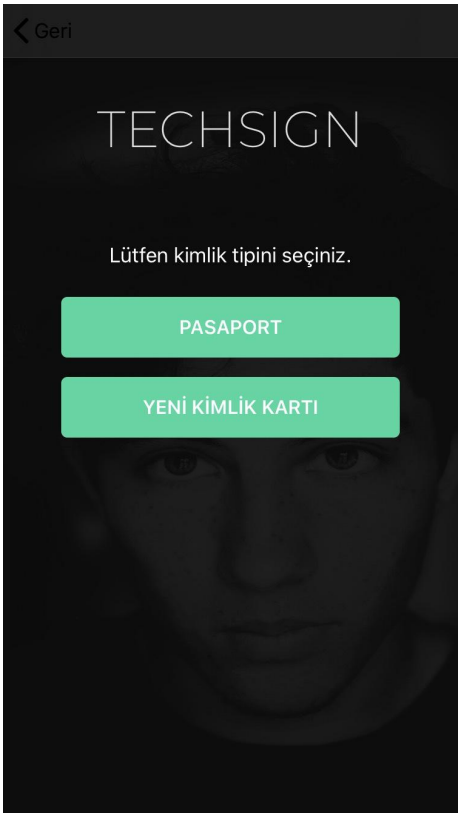
UIViewController

```
public override func viewWillAppear(_ animated: Bool) {
        AppUtility.lockOrientation(.portrait, andRotateTo: .portrait)
}

 public override func viewWillDisappear(_ animated: Bool) {
         //When controllers Disappear unlock all orientation for other controllers.
         AppUtility.lockOrientation(.all) // app's default
}
```

There are two methods for registering, OCR and NFC.

- **With NFC**



For the NFC case, we should use PASSPORT or NEW_TC_ID_BACK.

When the card photo is taken the card detection process starts. When the card detection process is finished, we receive a callback to *“cardDetected”.* Which we implemented above.
We can get the card image from this callback and convert it to base64 format;

let base64 = image!.pngData()!.base64EncodedString()

Then we make an API call for OCR. (ServerCall.verifyMrzOcr)

API Call:

Server Suffix: /id/mrz-ocr

Type: POST

Input:

{
img:String //Base64 of captured card image
idType: String // “PASSPORT” for a passport and “IDCARD” for an id card
}

Output:

MrzOcrOutputModel, details listed below;

id:String => Citizen Number (TCKN)
name:String => Name
surname:String => Surname
birthdate:String => Birth Date in YYMMDD format
expiredate:String => Expire Date in YYMMDD format
docNumber:String => Document Number

We need to check if the device has NFC.

```
if NFCNDEFReaderSession.readingAvailable, #available(iOS 13.0, *){
        //device has NFC
}else{
        //device has not NFC
}
```

NFC verification requires document number, birth date and expire date. All of these are in *"*MrzOcrOutputModel*"* from *"*ServerCall.verifyMrzOcr*)"*. Note that the dates of

*“*MrzOcrOutputModel*”* are in the form of DD.MM.YYYY. We need to convert them to YYMMDD form for the NFC verification.

Active Authentication (Optional)

For active authentication you should generate a nonce using ServerCall.generateNonce() which takes a transactionId as input. If you don’t have a transactionId yet, you can create one using ServerCall.startTransaction().

```
ServerCall.startTransaction(model: TransactionStartInputModel()) { (result) in
        switch(result){
           case .success(let transactionModel):
              ServerCall.generateNonce(transactionId: transactionModel.transactionId) { (res) in
                 switch(res){
                 case .success(let nonceModel):
                    print("nonce: " + nonceModel.nonce) // nonce will be used in NFCReader read()
function
                             see below
                    break
                 case .failure(let err):
                    break
                 }
              }
              break
           case .failure(let error):
              break
        }
     }
```

For NFC verification,

- First create a *LocalCommandTransmitter.*

  ```
  self.localCommandTransmitter = LocalNFCCommandTransmitter()
  ```

- Then create a *PassportReader* using it.

  ```
  self.passportReader = PassportReader(commandTransmitter: localCommandTransmitter!)
  ```

- Make your class implements *PassportReaderCallback.*

  ```
  func result(passportModel: PassportModel) {

        self.localCommandTransmitter?.showAlertMessage(alertMessage: “Passport read
  completed successfully”)

        // returns result when nfc operation is successful
  ```

```
            let validationInfo = IDValidationModel(dg1: passportModel.getDg1()!,
                                dg2: passportModel.getDg2()!,
                                sod: passportModel.getSod()!,
                                dg11: passportModel.getDg11(),
                                dg12: passportModel.getDg12(),
                                dg14: passportModel.getDg14(),
                                dg15: passportModel.getDg15(),
                                nonce: passportModel.getNonce(),
                                signedData: passportModel.getSignedData())
let validationModel = ValidationInputModel(validationInformation: validationInfo,
                                    idType: self.idType!)

 ServerCall.validateAndSaveID(model: validationModel) { (result) in
switch (result) {
case .success(let returnModel):
        // server call succeed
        break;
case .failure(let error):
        print("error on validateAndSaveID: \(error.description)")
        break;

}

func progress(progressType: NFCOperationType, progress: Double) {
        // returns the progress
        // you can show a custom message according to progress for example see below
        // for details see Customization part at the end of the document
        if progressType == NFCOperationType.DG1 {
                let progressString = self.handleProgress(percentualProgress: Int(progress *
100))
                message =  NSLocalizedString("reading", comment: "") + "
\("DG1").....\n\n\(progressString)"
              }

        self.localCommandTransmitter?.showAlertMessage(alertMessage: message)

}

 func error(error: Error) {
        // called if any error occurred in nfc progress
        // you can show a custom message according to error type for example see below

        switch error {

        // returns when any of docNo, birthDate and expireDate not match with the passport
        case TagError.InvalidMRZKey:
                print("InvalidMRZKey")
                errorMessage = "MRZ key is invalid check your inputs"
        }

        self.localCommandTransmitter?.invalidate(errorMessage: errorMessage)
```

}

- To start nfc verification.

self.passportReader.read(documentNumber: <DOCUMENT_NUMBER>, birthDate: <BIRTH_DATE>, expireDate: <EXPIRE_DATE>, nonce: <OPTIONAL_NONCE> , callback: self)

Then using the model "*ValidationInputModel*" for ServerCall.validateAndSaveID(), we can receive *"SaveIdReturnModel"* which consists of,

- String *"transactionId"* => will be used in liveness process
- String *"gestureType"* => which gesture should be used while recording the video "EYE" or "MOUTH".
- IDValidationResultModel "validationResult" => Consists of 5 boolean;

❏ boolean dg1valid
❏ boolean dg2valid
❏ boolean sodvalid
❏ boolean docSigningValid
❏ boolean acceptable
❏ boolean isNonceValid
❏ boolean dg15Valid
❏ booelan isAAValid

- [ControlEntryModel] “controlResults” => ControlEntryModel consists of,
    - String “entryType” => ControlEntryTypes enum[2]
    - Float “entryScore” => the result of the control between 0 and 1.
    - Boolean “entryInterpretation” => true if the control is successful
    - String “controlId” => unique identifier of the control

[2].public enum ControlEntryTypes : String{
    case PhotoCheatVerification = "PHOTOCHEAT"
    case HologramVerification = "HOLOGRAM"
    case HologramFaceVerification = "HOLOGRAMFACE"
    case FaceLivenessVerification = "FACELIVENESS"
    case FaceVerification = "FACE"
    case HiddenPhotoVerification = "HIDDENPHOTO"
    case SignaturePhotoVerification = "SIGNATUREPHOTO"
    case GuillocheVerification = "GUILLOCHEVERIFICATION"
    case RainbowVerification = "RAINBOWVERIFICATION"
    case OCRNFCSimilarity = "OCRNFCSIMILARITY"
    case OCRMRZSimilarity = "OCRMRZSIMILARITY"
}

API Call:

Server Suffix: /id/validate-and-save/

Type: POST

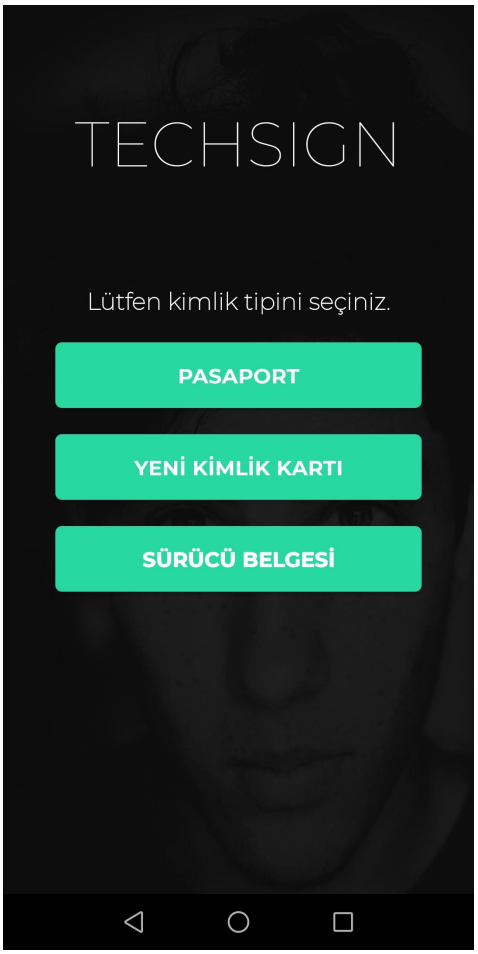
Input:

{
    validationInformation {
        dg1:String // base64 of dg1 file in NFC chip
        dg2:String // base64 of dg2 file in NFC chip
        sod:String // base64 of sod file in NFC chip
        dg11:String // base64 of dg11 file in NFC chip
        dg12:String // base64 of dg12 file in NFC chip
        dg14:String // base64 of dg14 file in NFC chip
        dg15:String // base64 of dg15 file in NFC chip
        nonce:String // base64 of active auth nonce (optional)
        signedData:String // base64 of active auth signed data (optional)
    }
    idType: String // “PASSPORT” for a passport and “IDCARD” for an id card
    transactionId:String // if the transaction started with ocr process before nfc process the transactionId of the ocr process otherwise null
}

Output:

SaveIDReturnModel, details listed above.

- **With OCR**



Set below configurations to the *"IDCardDetectionView".*

conf = IDCardDetectionConfiguration()
conf!.cardType = CARD_TYPE[1]
conf!.isMrzActive = false
conf!.skipOcr = true

[1]. The parameter *"CARD_TYPE"* is an enum from *"IDCardType".* Which consists of

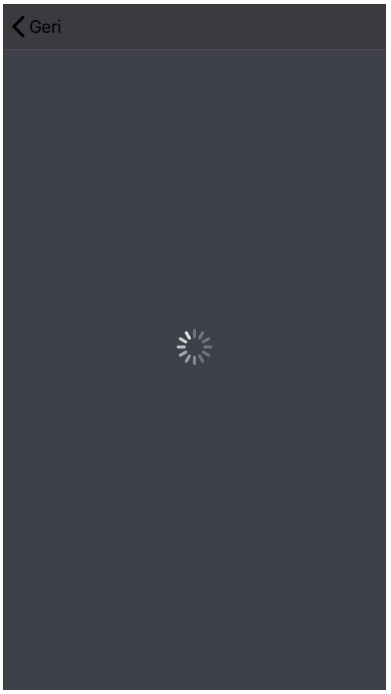
- UNKNOWN
- PASSPORT
- NEW_TC_ID
- OLD_TC_ID
- OLD_TC_DRIVER_LICENSE
- DRIVER_LICENSE
- NEW_TC_ID_BACK
- DRIVER_BACK

For the OCR case, we should use PASSPORT , NEW_TC_ID or DRIVER_LICENSE.

When the card image is captured the OCR process starts. When the card image is captured, we receive a callback to *“cardDetected”.* Which we implemented above. We can get the card image from this callback and convert it to base64 format;

```
let base64 =image!.pngData()!.base64EncodedString()
```

Then according to the *“CARD_TYPE”* we need to call ServerCall.verifyPassportPhoto(), ServerCall.verifyDriverPhoto() or ServerCall.verifyIDPhoto().



All of the calls require a CheckIDModel and a Result<CheckIdReturnModel,NSError>.

CheckIDModel contains a String which is the base64 of the captured UIImage above.

CheckIdReturnModel consists of,

- String “transactionId” => used for hologram control and liveness check.
- String “gestureType” => which gesture will be used for liveness check.
- [ControlEntryModel] “controlResults” => ControlEntryModel consists of,
    - String “entryType” => ControlEntryTypes enum[2]
    - Float “entryScore” => the result of the control between 0 and 1.
    - Boolean “entryInterpretation” => true if the control is successful
    - String “controlId” => unique identifier of the control

.

[2].public enum ControlEntryTypes : String{
	case PhotoCheatVerification = "PHOTOCHEAT"
	case HologramVerification = "HOLOGRAM"
	case HologramFaceVerification = "HOLOGRAMFACE"
	case FaceLivenessVerification = "FACELIVENESS"
	case FaceVerification = "FACE"
	case HiddenPhotoVerification = "HIDDENPHOTO"
	case SignaturePhotoVerification = "SIGNATUREPHOTO"
	case GuillocheVerification = "GUILLOCHEVERIFICATION"
	case RainbowVerification = "RAINBOWVERIFICATION"
	case OCRNFCSimilarity = "OCRNFCSIMILARITY"
	case OCRMRZSimilarity = "OCRMRZSIMILARITY"
}

Note: If the PhotoCheatVerification control in “controlResults” is successful we can continue to process. For the new id and new driver cards, the hologram process should be also completed before starting the liveness process.

API Call:

Server Suffix: /id/check/new-id
/id/check/new-driver
/id/check/passport

Type: POST

Input:

{
img:String //captured ID photo, in base64 format

transactionId:String // if the verification process started with nfc transaction id of nfc process else null

name:String // given name if not null OCR service calculates levenshtein distance and it is lower than %10 returns found name else returns given name, if no name present it should be null

surname:String // given surname if not null OCR service calculates levenshtein distance and it is lower than %10 returns found surname else returns given surname, if no surname present it should be null

expiredate:String // given expiredate if not null OCR service calculates levenshtein distance and it is lower than %10 returns found expiredate else returns given expiredate, if no expiredate present it should be null

birthdate:String // given birthdate if not null OCR service calculates levenshtein distance and it is lower than %10 returns found birthdate else returns given birthdate, if no birthdate present it should be null

}

Output:

CheckIdReturnModel, details listed above.

HOLOGRAM

First *'HologramDelegate'* should be implemented in a ViewController that will use *'HologramController'* .

```
public protocol HologramDelegate {
    func onVideoCaptured(videoURL:URL,rotate:Int) // video captured
    func onVideoStarted() // video recording started
    func onVideoCanceled() // video canceled
    func onFaceDetectionSuccess() // face detection succeed
    func onFaceDetectionIdle() // no face detected

}
```

Then *'HologramController'* should be added as a subview in the ViewController.

```
    var hologramController = HologramController()
    hologramController.setDelegate(self) // we set the delegate implemented before.
    livenessContainer.addSubview(hologramController.view) // add the hologramController view to another view (like a container, which has a adjusted position and frame) as subview
```

Note: When the controller's view added as a subview, the face detection process starts and when the user's face is detected properly, video recording immediately starts and the onVideoStarted*()* function is called. You can show a countdown timer on screen for a better user experience like in the demo application.

Note 2: Face detection layer (default layer is dashed rectangle) is configurable. See "*Customization"* section at the bottom of the document.

When *func onVideoCaptured(videoURL:URL,rotate:Int)* called video capture is finished. You can take the base64 of the video as below.

```
let fileData = try? Data(contentsOf: outputURL)
let base64 = fileData!.base64EncodedString()
```

ServerCall.detectHologram() function takes two parameters, a HologramDetectionInputModel and a TechsignServiceListener<HologramDetectionReturnModel>.

HologramDetectionInputModel consists of four fields,

- String “video” => base64 of recorded video[1]
- String “transactionId” => gathered from card process
- int “rotate” => rotation of recorded video (0,90,180 or 270)
- String “cardType” => “NEW-ID” for new id card, “NEW-DRIVER” for new drivers card.

HologramDetectionReturnModel consists of,

- ControlEntryModel[] “controlResults” => ControlEntryModel consists of,
  - String “entryType” => ControlEntryTypes enum[2]
  - Float “entryScore” => the result of the control between 0 and 1.
  - Boolean “entryInterpretation” => true if the control is successful
  - String “controlId” => unique identifier of the control

[1]. The recorded video should be around 5 seconds and 480p. The flash of the camera should be active after 1.5 second.

[2].
```
public enum ControlEntryTypes : String{
    case PhotoCheatVerification = "PHOTOCHEAT"
    case HologramVerification = "HOLOGRAM"
    case HologramFaceVerification = "HOLOGRAMFACE"
    case FaceLivenessVerification = "FACELIVENESS"
    case FaceVerification = "FACE"
    case HiddenPhotoVerification = "HIDDENPHOTO"
    case SignaturePhotoVerification = "SIGNATUREPHOTO"
    case GuillocheVerification = "GUILLOCHEVERIFICATION"
    case RainbowVerification = "RAINBOWVERIFICATION"
    case OCRNFCSimilarity = "OCRNFCSIMILARITY"
    case OCRMRZSimilarity = "OCRMRZSIMILARITY"
}
```

Note: For a successful hologram control both HologramFaceVerification and HologramVerification should be passed in “controlResults”.

API Call:

Server Suffix: /id/hologram

Type: POST

Input:

{
video: String //captured hologram video, in base64 format
rotate: Int //clock-wise rotation to apply to the video before control, 0/90/180/270
cardType:String //“NEW-ID” for new id card, “NEW-DRIVER” for new drivers card
transactionId:String //transaction ID for the on-going transaction
}

Output:

HologramDetectionReturnModel, details listed above.

Add Secondary Info (Optional)

You can add secondary info for NEW ID and NEW DRIVER cards. For example, if you started an ocr process with NEW ID card front side, you can add NEW ID back side and get the match result, also if you started the process with NEW ID mrz process you can also add the front side of the NEW ID and get the match result.

You can call *ServerCall.addSecondary()* to add secondary info.

API Call:

Server Suffix: /id/add-secondary

Type: POST

Input:

{
    image:String // base64 of captured card image
    transactionID:String
}

Output:

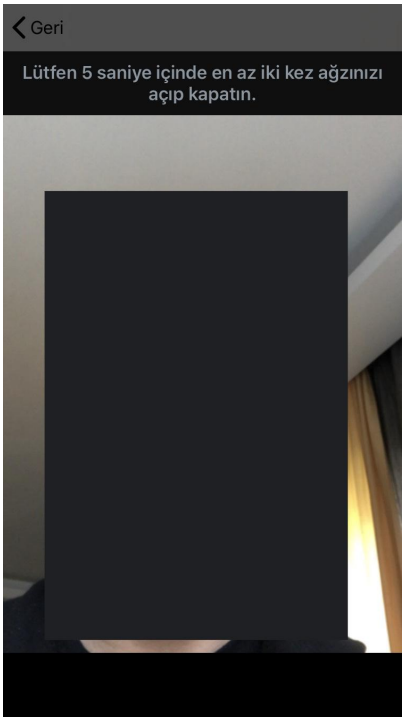
{
matchResult:Boolean // true if both side of the cards matched
controlResults:[ControlEntryModel][1] // check *OCRNFCSimilarity* and *OCRMRZSimilarity*
}

[1] ControlEntryModel[] “controlResults” => ControlEntryModel consists of,

- String “entryType” => ControlEntryTypes enum[2]
- Float “entryScore” => the result of the control between 0 and 1.
- Boolean “entryInterpretation” => true if the control is successful
- String “controlId” => unique identifier of the control

[2].public enum ControlEntryTypes : String{
case PhotoCheatVerification = "PHOTOCHEAT"
case HologramVerification = "HOLOGRAM"
case HologramFaceVerification = "HOLOGRAMFACE"
case FaceLivenessVerification = "FACELIVENESS"
case FaceVerification = "FACE"
case HiddenPhotoVerification = "HIDDENPHOTO"
case SignaturePhotoVerification = "SIGNATUREPHOTO"
case GuillocheVerification = "GUILLOCHEVERIFICATION"
case RainbowVerification = "RAINBOWVERIFICATION"
case OCRNFCSimilarity = "OCRNFCSIMILARITY"
case OCRMRZSimilarity = "OCRMRZSIMILARITY"
}

B. <u>Liveness Process</u>



First *‘LivenessDelegate’* should be implemented in a ViewController that will use *‘LivenessController’* .

```
public protocol LivenessDelegate {
    func onVideoCaptured(videoURL:URL,rotate:Int) // video captured
    func onVideoStarting() //preparation time starting see VIDEO_PREPARATION_TIME_IN_SECONDS
    func onVideoStarted() // video recording started
    func onVideoCanceled() // video canceled
    func onFaceDetectionFailure(isAway: Bool) // face detection succeed but distance is far away if isAway is true, too close if isAway is false
    func onFaceDetectionSuccess() // face detection succeed and distance is proper
    func onFaceDetectionIdle() // no face detected
    func onMultiFaceDetected() // multiface detected

    // Gesture events
    func positionSucceed(model:PositionSuccedModel) // called when a position (left, up etc.) successfully completed, you may guide user for next position

    func positionFailed(model:PositionFailedModel) // called when a position (left, up etc.) failed, you may guide user to turn his/her head to required direction

    func positionCorrect() // called when user’s head is in correct direction, may be used to clear the guide message

    func turnedTooMuch(model:TurnedTooMuchModel) // called when the head degree is too much for detection, you may guide the user to make a slight head movement
```

func gestureFailed() // called after the recording completed but the required gesture is not completed, you may show the guidance again and restart the process

}

Then *'LivenessController'* should be added as a subview in the ViewController.

var livenessController = LivenessController()
livenessController.setDelegate(self) // we set the delegate implemented before.
livenessContainer.addSubview(livenessController.view) // add the livenessController's view to another view (like a container, which has a adjusted position and frame) as subview

Note: When the start() method called face detection process starts and when the user's face is detected properly, preparation time starts and *onVideoStarting()* function is called. You can show a countdown timer on screen for a better user experience like in the demo application. If no face detected, multiface detected or distance is not proper relevant delegate function is called and preparation time resets. If the user's face is properly detected upon the preparation time, video recording starts and *onVideoStarted()* function is called.

Note 2: Face detection layer (default layer is dashed rectangle) is configurable. See "*Customization"* section at the bottom of the document.

When *func onVideoCaptured(videoURL:URL,rotate:Int)* called video capture is finished. You can take the base64 of the video as below.

```
let fileData = try? Data(contentsOf: outputURL)
let base64 = fileData!.base64EncodedString()
```

Liveness service can be used by calling ServerCall.checkLiveness() function which requires a FaceVerificationModel and a Result<VerifyFaceReturnModel,NSError>.

FaceVerificationModel consists of three fields,

- String "video" => base64 of recorded selfie video[1]
- String "transactionId" => gathered from card process
- int "rotate" => rotation of recorded video (0,90,180 or 270)

[1]. The recorded video should be around 5 seconds and 480p.

If the service call is successful, onSuccess method called with a VerifyFaceReturnModel which consist of,

- String "id" => citizen id
- String "name" => user's name
- String "surname" => user's surname

- String “birthDate” => user’s birth date
- String “expireDate” => user’s expire date
- String “picture” => base64 of user’s picture on identity card which he/she used for registration
- [ControlEntryModel] “controlResults” => ControlEntryModel consists of,
    - String “entryType” => ControlEntryTypes enum[2]
    - Float “entryScore” => the result of the control between 0 and 1.
    - Boolean “entryInterpretation” => true if the control is successful
    - String “controlId” => unique identifier of the control

[2].public enum ControlEntryTypes : String{
    case PhotoCheatVerification = "PHOTOCHEAT"
    case HologramVerification = "HOLOGRAM"
    case HologramFaceVerification = "HOLOGRAMFACE"
    case FaceLivenessVerification = "FACELIVENESS"
    case FaceVerification = "FACE"
    case HiddenPhotoVerification = "HIDDENPHOTO"
    case SignaturePhotoVerification = "SIGNATUREPHOTO"
    case GuillocheVerification = "GUILLOCHEVERIFICATION"
    case RainbowVerification = "RAINBOWVERIFICATION"
    case OCRNFCSimilarity = "OCRNFCSIMILARITY"
    case OCRMRZSimilarity = "OCRMRZSIMILARITY"
}

Note: For a successful register both *“faceResult”* and *“livenessResult”* should be true.

API Call:

Server Suffix: /face/verify

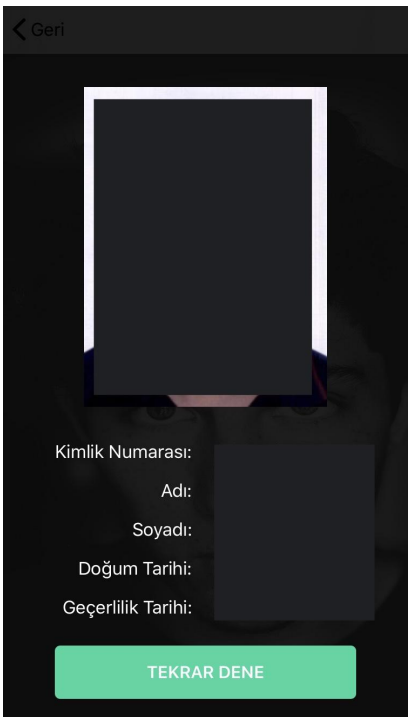
Type: POST

Input:

{
transactionId:String //transaction ID for the on-going transaction
video: String //captured liveness video, in Base64 format
rotate: Int //clock-wise rotation to apply to the video before control, 0/90/180/270
}

Output:

VerifyFaceReturnModel, details listed above.



We use AVFoundation api on the sample app(RKYC-iOS-Test-App) for video recording,
see https://stackoverflow.com/a/41698917/4824165

Finalize Transaction (Optional)

Finalize transaction service takes the *transactionId* and makes its job ,if there is any, then returns the *result* and a *message.*

You can finalize the transaction by calling *ServerCall.finalizeTransaction()*

API Call:

Server Suffix: /id/finalize-transaction

Type: POST

Input:

```
{
        transactionId:String
}
```

Output:

{
    result:Boolean // true if finalization succeed
    message:String

}

Customization

OCR

```
 conf = IDCardDetectionWrapperConfiguration()

 /*
 common configurations
*/

// card type list should be given
// default value is  [.NEW_ID, .NEW_ID_BACK, .NEW_DRIVER, .DRIVER_BACK]
// should give same size cards for below ios13, otherwise throws fatal error

conf.cardTypeList = cardTypes

// hint stroke width
// default value is 10.0

conf.hintStrokeWidth = 10.0

// color of hint
// default value is UIColor.blue.cgColor

conf.hintColor = UIColor.blue.cgColor

// color of inside hint area
// default value is UIColor.init(red: 1.0, green: 1.0, blue: 1.0, alpha: 0.0)

conf.hintBackgroundColor = UIColor.init(red: 1.0, green: 1.0, blue: 1.0, alpha: 0.0)

// color of outside hint area
// default value is UIColor.init(red: 1.0, green: 1.0, blue: 1.0, alpha: 0.5)

conf.overlayBackgroundColor = UIColor.init(red: 1.0, green: 1.0, blue: 1.0, alpha: 0.5)


// hint size over view size according to restrictive side
// should between 0-1
// default value is 0.75

conf.hintDisplayRatio = 0.75
```

```
/*
 cnn - ios13+ configurations
 */

// hides hint, we don't need it
// default value is true

conf.isHintHidden = true

// color of captured card
// default value is UIColor.init(red: 0.0, green: 1.0, blue: 0.0, alpha: 0.25).cgColor

conf.capturedColor = UIColor.init(red: 0.0, green: 1.0, blue: 0.0, alpha: 0.25).cgColor

// color of uncaptured card
// default value is UIColor.init(red: 1.0, green: 0.0, blue: 0.0, alpha: 0.25).cgColor

conf.uncapturedColor = UIColor.init(red: 1.0, green: 0.0, blue: 0.0, alpha: 0.25).cgColor

// how to draw captured/uncaptured cards
// fill the rectangle or only stroke the edges
// default value is .fillStrokne

conf.capturedStyle = .fillStroke

// captured stroke width
// effective when capturedStyle set to .stroke
// default value is 10.0

conf.capturedStrokeWidth = 10.0

// check card type using ocr
// we check card type using ml model  dynmically
// default value is false

conf.checkCardType = false

// correct rotation if reversen taken card detected
// active only check card type set to true
// default value is false

conf.turnIfReverse = false


/*
 below13
*/


// color of hint edges when card captured on that hint
// default value is UIColor.blue.cgColor
```

```
conf.capturedHintColor = UIColor.blue.cgColor

// hint edges type
// dashed line or default line
// default value is .DEFAULT

conf.captureType = .DEFAULT

// id photo hint
// draws another hint for id photo
// default value is false

conf.idPhotoHint = false

// ratio of hint edges over undetected card edges
// default value is 4.0

conf.preHintRatio = 4.0
```

NFC

You can customize the NFC reading screen by showing custom messages. Unfortunately the iOS SDK doesn't let us change the device animation and the label above the screen ("Ready to Scan" for English, "Tarama için Hazır" for Turkish) We can only set the message below.

1- For the initial message;

```
LocalNFCCommandTransmitter.NFC_START_MESSAGE = "Hold your iPhone near an NFC enabled passport."
```

2- By using the PassportCallback's progress function.

```
func progress(progressType: NFCOperationType, progress: Double) {
    var message = ""
    if progressType == NFCOperationType.AUTHENTICATION {
        let progressString = self.handleProgress(percentualProgress: Int(progress * 100))
        message = "Authenticating with passport.....\n\n\(progressString)"
    }
    else if progressType == NFCOperationType.DG1 {
        let progressString = self.handleProgress(percentualProgress: Int(progress * 100))
        message =  NSLocalizedString("reading", comment: "") + " \("DG1").....\n\n\(progressString)"
    }
    else if progressType == NFCOperationType.DG2 {
```

```
            let progressString = self.handleProgress(percentualProgress: Int(progress * 100))
            message =  NSLocalizedString("reading", comment: "") + " \("DG2").....\n\n\(progressString)"
        }
        else if progressType == NFCOperationType.DG11 {
            let progressString = self.handleProgress(percentualProgress: Int(progress * 100))
            message =  NSLocalizedString("reading", comment: "") + " \("DG11").....\n\n\(progressString)"
        }
        else if progressType == NFCOperationType.DG12 {
            let progressString = self.handleProgress(percentualProgress: Int(progress * 100))
            message =  NSLocalizedString("reading", comment: "") + " \("DG12").....\n\n\(progressString)"
        }
        else if progressType == NFCOperationType.DG14 {
            let progressString = self.handleProgress(percentualProgress: Int(progress * 100))
            message =  NSLocalizedString("reading", comment: "") + " \("DG14").....\n\n\(progressString)"
        }
        else if progressType == NFCOperationType.DG15 {
            let progressString = self.handleProgress(percentualProgress: Int(progress * 100))
            message =  NSLocalizedString("reading", comment: "") + " \("DG15").....\n\n\(progressString)"
        }
        else if progressType == NFCOperationType.SOD {
            let progressString = self.handleProgress(percentualProgress: Int(progress * 100))
            message =  NSLocalizedString("reading", comment: "") + " \("SOD").....\n\n\(progressString)"
        }


         self.localCommandTransmitter?.showAlertMessage(alertMessage: message)



    }

    func handleProgress(percentualProgress: Int) -> String {
        let p = (percentualProgress/20)
        let full = String(repeating: "🟢 ", count: p)
        let empty = String(repeating: "⚪️ ", count: 5-p)
        return "\(full)\(empty)"
    }
```

3- By using the PassportCallback's error function.

```
func error(error: Error) {

        var errorMessage = ""

        switch error {

        case TagError.ResponseError(let error, let code1, let code2):
            print("ResponseError: err: \(error) code1: \(code1) code2: \(code2)")
            errorMessage = "Sorry, there was a problem reading the passport. Please try again"
        case TagError.InvalidResponse:
            print("InvalidResponse")
            errorMessage = "Sorry, there was a problem reading the passport. Please try again"
```

```
case TagError.UnexpectedError:
    print("UnexpectedError")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.NFCNotSupported:
    print("NFCNotSupported")
    errorMessage = "NFC Not Supported for this device"
case TagError.NoConnectedTag:
    print("NoConnectedTag")
    errorMessage = "Please keep and hold your passport near NFC area"
case TagError.D087Malformed:
    print("D087Malformed")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.InvalidResponseChecksum:
    print("InvalidResponseChecksum")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.MissingMandatoryFields:
    print("MissingMandatoryFields")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.CannotDecodeASN1Length:
    print("CannotDecodeASN1Length")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.InvalidASN1Value:
    print("InvalidASN1Value")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.UnableToProtectAPDU:
    print("UnableToProtectAPDU")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.UnableToUnprotectAPDU:
    print("UnableToUnprotectAPDU")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.UnsupportedDataGroup:
    print("UnsupportedDataGroup")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.DataGroupNotRead:
    print("DataGroupNotRead")
    errorMessage = "DataGroupNotRead"
case TagError.UnknownTag:
    print("UnknownTag")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.UnknownImageFormat:
    print("UnknownImageFormat")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.NotImplemented:
    print("NotImplemented")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.TagNotValid:
    print("TagNotValid")
    errorMessage = "Sorry, there was a problem reading the passport. Please try again"
case TagError.ConnectionError:
    print("ConnectionError")
    errorMessage = "Please keep and hold your passport near NFC area"
case TagError.UserCanceled:
```

```
            // user clicked to cancel (“vazgeç” in Turkish) button
            print("UserCanceled")
            errorMessage = "User Canceled"
        case TagError.InvalidMRZKey:
            // at least one of the three inputs (docNo, birthDate, expireDate) doesn’t match with the passport
            print("InvalidMRZKey")
            errorMessage = "MRZ Key not valid for this document."
        case TagError.MoreThanOneTagFound:
            print("MoreThanOneTagFound")
            errorMessage = "More than 1 tags was found. Please present only 1 tag."
        case TagError.InvalidHashAlgorithmSpecified:
            print("InvalidHashAlgorithmSpecified")
            errorMessage = "Sorry, there was a problem reading the passport. Please try again"
        case TagError.InvalidDataPassed(let error):
            print("InvalidDataPassed: \(error)")
            errorMessage = "Sorry, there was a problem reading the passport. Please try again"
        case TagError.SessionTimeout:
            // nfc session timeouts after 60 seconds.
            print("SessionTimeout")
            errorMessage = "Session Timeout"
        case TagError.AuthenticationFailed:
            print("AuthenticationFailed")
            errorMessage = "Please remove phone case and phone cover if any, keep and hold your passport near NFC area"
        case TagError.ActiveAuthNotSupportedForPassport:
            // the passport does not support active auth, you can try with nil nonce
            print("ActiveAuthNotSupportedForPassport")
            errorMessage = "ActiveAuthNotSupportedForPassport"


        }

        self.localCommandTransmitter?.invalidate(errorMessage: errorMessage)

    }
```

Liveness

You can change the default face detection layers shape like in the below.

LivenessController.FACE_DETECTION_IDLE_LAYER = <CUSTOM FACE DETECTION IDLE LAYER>
LivenessController.FACE_DETECTION_SUCCESS_LAYER = <CUSTOM FACE DETECTION SUCCESS LAYER>
LivenessController.FACE_DETECTION_FAIL_LAYER = <CUSTOM FACE DETECTION FAIL LAYER>

Default layers are in the below.

```
private static func createDefaultIdleLayer() -> CAShapeLayer {
        let layer = CAShapeLayer()
        layer.strokeColor = UIColor.white.cgColor
        layer.lineDashPattern = [100, 100]
        layer.fillColor = nil
        return layer
}
private static func createDefaultFailLayer() -> CAShapeLayer {
        let layer = CAShapeLayer()
        layer.strokeColor = UIColor.red.cgColor
        layer.lineDashPattern = [100, 100]
        layer.fillColor = nil
        return layer
}
private static func createDefaultSuccessLayer() -> CAShapeLayer {
        let layer = CAShapeLayer()
        layer.strokeColor = UIColor.green.cgColor
        layer.lineDashPattern = [100, 100]
        layer.fillColor = nil
        return layer
}
```

Hologram

You can change the default face detection layers shape like in the below.

```
HologramController.FACE_DETECTION_IDLE_LAYER = <CUSTOM FACE DETECTION IDLE LAYER>
HologramController.FACE_DETECTION_SUCCESS_LAYER = <CUSTOM FACE DETECTION SUCCESS LAYER>
```

Default layers are in the below.

```
private static func createDefaultIdleLayer() -> CAShapeLayer {
        let layer = CAShapeLayer()
        layer.strokeColor = UIColor.white.cgColor
        layer.lineDashPattern = [100, 100]
        layer.fillColor = nil
        return layer
}

private static func createDefaultSuccessLayer() -> CAShapeLayer {
        let layer = CAShapeLayer()
        layer.strokeColor = UIColor.green.cgColor
        layer.lineDashPattern = [100, 100]
        layer.fillColor = nil
        return layer
}
```