class ViewController: UIViewController, CLLocationManagerDelegate {

var mapView: MKMapView!

var locationManager: CLLocationManager!

override func viewDidLoad() {

super.viewDidLoad()

// Setup mapView

mapView = MKMapView(frame: self.view.bounds)

self.view.addSubview(mapView)

// Setup locationManager

locationManager = CLLocationManager()

locationManager.delegate = self

locationManager.requestWhenInUseAuthorization()

// Start updating location

locationManager.startUpdatingLocation()

}

func locationManager(\_ manager: CLLocationManager,

didUpdateLocations locations: [CLLocation]) {

if let location = locations.last {

// Center the map on the user's current location

let region = MKCoordinateRegion(center: location.coordinate,

latitudinalMeters: 500,

longitudinalMeters: 500)

mapView.setRegion(region, animated: true)

}

}

}

NotificationCenter.default.addObserver( self,

selector: #selector(accessoryDidConnect), name: .EAAccessoryDidConnect,

object: nil

)

NotificationCenter.default.addObserver(self,

selector: #selector(accessoryDidDisconnect), name: .EAAccessoryDidDisconnect,

object: nil

)

EAAccessoryManager.shared().registerForLocalNotifications()

var session: EASession?

var accessory: EAAccessory?

func startSession() {

guard let accessory = EAAccessoryManager.shared().connectedAccessories.first else {

print("No connected accessories")

return

}

guard let protocolString = accessory.protocolStrings.first else {

print("No supported protocol strings")

return

}

session = EASession(accessory: accessory, forProtocol: protocolString)

if let session = session {

session.outputStream?.delegate = self

session.outputStream?.schedule(in: .current, forMode: .common)

session.outputStream?.open()

session.inputStream?.delegate = self

session.inputStream?.schedule(in: .current, forMode: .common)

session.inputStream?.open()

}

}

let pipelineDescriptor = MTLRenderPipelineDescriptor()

pipelineDescriptor.vertexFunction = vertexShader

pipelineDescriptor.fragmentFunction = fragmentShader

pipelineDescriptor.colorAttachments[0].pixelFormat = .bgra8Unorm

let renderPipelineState = try device.makeRenderPipelineState(descriptor: pipelineDescriptor)

guard let commandBuffer = commandQueue.makeCommandBuffer(),

let renderPassDescriptor = view.currentRenderPassDescriptor,

let renderEncoder = commandBuffer.makeRenderCommandEncoder(descriptor: renderPassDescriptor) else {

fatalError("Failed to create Metal command encoder")

}

renderEncoder.setRenderPipelineState(renderPipelineState)

// Set other render configurations, like vertex buffers and draw calls

renderEncoder.drawPrimitives(type: .triangle, vertexStart: 0,

vertexCount: vertices.count)

renderEncoder.endEncoding()

commandBuffer.present(view.currentDrawable!)

commandBuffer.commit()

let image = UIImage(named: "someImage")

let request = VNDetectFaceRectanglesRequest { request, error in

guard let results = request.results as? [VNFaceObservation] else { return }

for face in results {

print("Found face at \(face.boundingBox)")

}

}

if let cgImage = image?.cgImage {

let handler = VNImageRequestHandler(cgImage: cgImage, options: [:])

try? handler.perform([request])

}

let textRequest = VNRecognizeTextRequest { request, error in

guard let observations = request.results as? [VNRecognizedTextObservation] else { return }

for observation in observations {

if let topCandidate = observation.topCandidates(1).first {

print("Recognized text: \(topCandidate.string)")

}

}

}

try? handler.perform([textRequest])

import SafariServices

func presentWebPage(url: URL) {

let safariViewController = SFSafariViewController(url: url)

present(safariViewController, animated: true)

}

import AuthenticationServices

var authSession: ASWebAuthenticationSession?

func authenticateUser(url: URL, callbackURLScheme: String, completionHandler: @escaping (URL?, Error?) -> Void) {

authSession = ASWebAuthenticationSession(url: url,

callbackURLScheme: callbackURLScheme,

completionHandler: completionHandler)

authSession?.start()

}

extension MyViewController: SFSafariViewControllerDelegate {

func safariViewControllerDidFinish(\_ controller: SFSafariViewController) {

print("SafariViewController finished")

}

}

if PKPaymentAuthorizationViewController.canMakePayments() {

// Apple Pay is available, and the user has added at least one payment card

}

let paymentRequest = PKPaymentRequest()

paymentRequest.merchantIdentifier = "your.merchant.id"

paymentRequest.supportedNetworks = [.visa, .masterCard, .amex]

paymentRequest.merchantCapabilities = .capability3DS

paymentRequest.countryCode = "US"

paymentRequest.currencyCode = "USD"

paymentRequest.paymentSummaryItems = [

PKPaymentSummaryItem(label: "Item 1", amount: NSDecimalNumber(string: "10.00")),

PKPaymentSummaryItem(label: "Total", amount: NSDecimalNumber(string: "10.00"))

]

if let paymentVC = PKPaymentAuthorizationViewController(paymentRequest: paymentRequest) {

paymentVC.delegate = self

present(paymentVC, animated: true)

}

extension YourViewController: PKPaymentAuthorizationViewControllerDelegate {

func paymentAuthorizationViewController(\_ controller: PKPaymentAuthorizationViewController, didAuthorizePayment payment: PKPayment, completion: @escaping (PKPaymentAuthorizationStatus) -> Void) {

// Process the payment, e.g., send it to your payment provider

completion(.success)

}

func paymentAuthorizationViewControllerDidFinish(\_ controller: PKPaymentAuthorizationViewController) {

// Dismiss the payment authorization view controller

controller.dismiss(animated: true, completion: nil)

}

}

import FBSDKLoginKit

let loginManager = LoginManager()

loginManager.logIn(permissions: ["public\_profile", "email"], from: self) { (result, error) in

if let error = error {

print("Failed to login: \(error.localizedDescription)")

return

}

// Handle login success ...

}

import FBSDKShareKit

let content = ShareLinkContent()

content.contentURL = URL(string: "https://example.com")!

ShareDialog(fromViewController: self, content: content, delegate: nil).show()

import LinkedInSDK

let linkedInManager = LinkedInManager()

linkedInManager.authenticate(from: self) { (success, error) in

if success {

// Fetch user profile data

}

}