install cocoapods ⇒ $ sudo gem install cocoapods

Ruby command, mac already come with Ruby

```
pod 'Firebase/Core'
pod 'Firebase/Auth'
pod 'Firebase/Database
pod 'RevealingSplashView'
```

bottom layer
top layer

//Visual effect view with Blurr ≤ add the view in storyboard

//CA Gradient layer

//AppDelegate.

p-list ⇒ always use locatio

```
import UIKit
import Firebase

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate{
    var window: UIWindow?

    fileprivate var containerVC = ContainerVC()

    var MenuContainerVC: ContainerVC{
        return containerVC
    }

    func application(_ application: UIApplication, didFinishLaunchingWithOptions LaunchOptions:
        [UIApplication.launchOptionsKey: Any]?) -> Bool{
        FIRApp.configure()
        containVC = ContainerVC()

        window?.rootViewController = containerVC
        window?.makeKeyAndVisible()

        return true
    }

    class func getAppDelegate() -> AppDelegate{
        return UIApplication.shared.delegate as! AppDelegate
    }

}
```

4-4

// DataService.swift (services)

```
import Foundation
import Firebase

let DB_BASE = FIRDatabase.database().reference()

class DataService {
    static let instance = DataService()

    private var _REF_BASE = DB_BASE
    private var _REF_USERS = DB_BASE.child("users")
    private var _REF_DRIVERS = DB_BASE.child("drivers")
    private var _REF_TRIPS = DB_BASE.child("trips")

    var REF_BASE: FIRDatabaseReference {
        return _REF_BASE
    }

    var REF_USERS: FIRDatabaseReference {
        return _REF_USERS
    }

    var REF_DRIVERS: FIRDatabaseReference {
        return _REF_DRIVERS
    }

    var REF_TRIPS: FIRDatabaseReference {
        return _REF_TRIPS
    }

    func createFirebaseDBUser(uid: String, userData: Dictionary<String, Any>, isDriver: Bool) {
        if isDriver {
            REF_DRIVERS.child(uid).updateChildValues(userData)
        } else {
            REF_USERS.child(uid).updateChildValues(userData)
        }
    }
```

// UIViewExt.swift (Extensions)

```
import UIKit

extension UIView{
    func fadeTo(alphaValue: CGFloat, withDuration duration: TimeInterval){
        UIView.animate(withDuration: duration){
            self.alpha = alphaValue
        }
    }
```

```
func bindtoKeyboard() {
    NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillChange(_:)), name: NSNotification.Name.UIKeyboardWillChangeFrame, object: nil)
}

func keyboardWillChange(_ notification: NSNotification) {
    let duration = notification.userInfo![UIKeyboardAnimationDurationUserInfoKey] as! Double
    let curve = notification.userInfo![UIKeyboardAnimationCurveUserInfoKey] as! UInt
    let curFrame = (notification.userInfo![UIKeyboardFrameBeginUserInfoKey] as! NSValue).cgRectValue
    let targetFrame = (notification.userInfo![UIKeyboardFrameEndUserInfoKey] as! NSValue).cgRectValue
    let deltaY = targetFrame.origin.y - curFrame.origin.y

    UIView.animateKeyframes(withDuration: duration, delay: 0.0, options: UIViewKeyframeAnimationOptions(rawValue: curve), animations: {
        self.frame.origin.y += deltaY
    }, completion: nil)
}
```

```
}
```

// CenterVCDelegate.swift (Protocols)

```
import UIKit

protocol CenterVCDelegate {
    func toggleLeftPanel()
    func addLeftPanelViewController()
    func animateLeftPanel(shouldExpand: Bool)
}
```

// GradientView.swift (View)

```
import UIKit

class GradientView : UIView{
    let gradient = CAGradientLayer()

    override func awakeFromNib(){
        setupGradientView()
    }

    func setupGradientView(){
        gradient.frame = self.bounds
        gradient.colors = [UIColor.white.cgColor, UIColor.init(white:1.0, alpha:0.0).cgColor]
        gradient.startPoint = CGPoint.zero   // top left corner
        gradient.endPoint = CGPoint(x:0, y:1)   //100%
        gradient.locations = [0.8, 1.0]  //where the gradient stops
        self.layer.addSublayer(gradient)
    }
}
```

(0,0) → x, y ↓

// set the back ground color to default

// RoundImageView (view)

```
import UIKit

class RoundImageView : UIImageView{

    override func awakeFromNib(){
        setupView()
    }

    func setupView(){
        self.layer.cornerRadius = self.frame.width/2
        self.clipsToBounds = true
    }
}
```

// RoundedShadowView.swift (View)

```
import UIKit

class RoundedShadowView: UIView{
    override func awakeFromNib(){
        setupView()
    }
    func setupView(){
        self.layer.cornerRadius = 5.0
        self.layer.shadowOpacity = 0.3
        self.layer.shadowColor = UIColor.darkGray.cgColor
        self.layer.shadowRadius = 5.0
        self.layer.shadowOffset = CGSize(width:0, height:5)
    }
}
```

// CircleView.swift (view)

```
import UIKit

class CircleView: UIView{
    @IBInspectable var borderColor: UIColor?{
        didSet{
            setupView()
        }
    }
    override func awakeFromNib(){
        setupView()
    }
    func setupView(){
        self.layer.cornerRadius = self.frame.width/2
        self.layer.borderWidth = 1.5
        self.layer.borderColor = borderColor?.cgColor
    }
}
```

// RoundedShadowButton.swift (view)

```
import UIKit

class RoundedShadowButton: UIButton{
    var originalSize: CGRect?

    override func awakeFromNib(){
        setupView()
    }

    func setupView(){
        originalSize = self.frame
        self.layer.cornerRadius = 5.0
        self.layer.shadowOpacity = 10.0
        self.layer.shadowColor = UIColor.darkGray.cgColor
        self.layer.shadowRadius = 0.3
        self.layer.shadowOffset = CGSize.zero
    }


    func animateButton(shouldLoad: Bool, withMessage message: String?){
        let spinner = UIActivityIndicatorView()
        spinner.activityIndicatorViewStyle = .whiteLarge
        spinner.color = UIColor.darkGray
        spinner.alpha = 0.0
        spinner.hidesWhenStopped = true
        spinner.tag = 21 // random tag
```

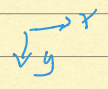
add the button to an empty view

```
        if shouldLoad{
            self.addSubview(spinner)
            self.setTitle("", for: .normal)
            UIView.animate(withDuration: 0.2, animations: {
                self.layer.cornerRadius = self.frame.height/2
                self.frame = CGRect(x: self.frame.midX - (self.frame.height/2), y: self.frame.origin.y,
                    width: self.frame.height, height: self.frame.height)
            }, completion: { (finished) in
                if finished == true {
                    spinner.startAnimating()
                    spinner.center = CGPoint(x: self.frame.width/2 + 1, y: self.frame.width/2 + 1)
                    spinner.fadeTo(alphaValue: 1.0, withDuration: 0.2)

                } // if
            }) // animate
            self.isUserInteractionEnabled = false
        } else {
            self.isUserInteractionEnabled = true
            for subview in self.subViews{
                if subview.tag == 21{
                    subview.removeFromSuperView()
                }
            }
            UIView.animate(withDuration: 0.2, animations: {
                self.layer.cornerRadius = 5.0
                self.frame = self.originalSize!
                self.setTitle(message, for: .normal)
            })
        }
    }
}
```

// RoundedCornerTextField.swift (View)

```
import UIKit
class RoundedCornerTextField: UITextField{
    var textRectOffset: CGFloat = 20
    override func awakeFromNib(){
        setupView()
    }
    func setupView(){
        self.layer.cornerRadius = self.frame.height/2
    }
    override func textRect(forBounds bounds: CGRect) -> CGRect{
        return CGRect(x: 0 + textRectOffset, y: 0 + (textRectOffset/2), width: self.frame.width -
            textRectOffset, height: self.frame.height + textRectOffset)
    }
    override func editingRect(forBounds bounds: CGRect) -> CGRect{
        return CGRect(x: 0 + textRectOffset, y: 0 + (textRectOffset/2), width: self.frame.width -
            textRectOffset, height: self.frame.height + textRectOffset)
    }
}
```



// ContainerVC.swift (Controller)

```
import UIKit
import QuartzCore

enum SlideOutState{
    case collapsed
    case leftPanelExpanded
}

enum ShowWhichVC{
    case homeVC
}
var showVC: ShowWhichVC = .homeVC

class ContainerVC: UIViewController{
    var homeVC: HomeVC!
    var leftVC: LeftSidePanelVC!
    var centerController: UIViewController!
    var currentState: SlideOutState = .collapsed{
        didSet{
            let shouldShowShadow = (currentState != .collapsed)
            shouldShowShadowForCenterViewController(status: shouldShowShadow)
        }
    }

    var isHidden = false
    let centerPanelExpandedOffset: CGFloat = 160   ← test and try
    var tap: UITapGestureRecognizer!

    override func viewDidLoad(){
        super.viewDidLoad()
        initCenter(screen: showVC)
    }
    func initCenter(screen: ShowWhichVC){
        var presentingController: UIViewController
        showVC = screen
        if homeVC == nil{
            homeVC = UIStoryboard.homeVC()
            homeVC.delegate = self
        }
        presentingController = homeVC
        if let con = centerController{
            con.view.removeFromSuperview()
            con.removeFromParentViewController()
        }
        centerController = presentingController

        view.addSubview(centerController.view)
        addChildViewController(centerController)
        centerController.didMove(toParentViewController: self)
```

```
    override var preferredStatusBarUpdateAnimation: UIStatusBarAnimation {
        return UIStatusBarAnimation.slide
    }

    override var prefersStatusBarHidden: Bool {
        return isHidden
    }
}

extension ContainerVC: CenterVCDelegate {
    func toggleLeftPanel() {
        let notAlreadyExpanded = (currentState != .leftPanelExpanded)

        if notAlreadyExpanded {
            addLeftPanelViewController()
        }
        animateLeftPanel(shouldExpand: notAlreadyExpanded)
    }
    func addLeftPanelViewController() {
        if leftVC == nil {
            leftVC = UIStoryboard.leftViewController()
            addChildSidePanelViewController(leftVC!)
        }
    }

    func addChildSidePanelViewController(_ sidePanelController: LeftSidePanelVC) {
        view.insertSubview(sidePanelController.view, at: 0)
        addChildViewController(sidePanelController)
        sidePanelController.didMoveToParentViewController(self)
    }

    func animateLeftPanel(shouldExpand: Bool) {
        if shouldExpand {
            isHidden = !isHidden
            animateStatusBar()
            setupWhiteCoverView()
            currentState = .leftPanelExpanded

            animateCenterPanelXPosition(targetPosition: centerController.view.frame.width - centerPanelExpandedOffset)
        } else {
            isHidden = !isHidden
            animateStatusBar()
            hideWhiteCoverView()
            animateCenterPanelXPosition(targetPosition: 0, completion: { (finished) in
                if finished == true {
                    self.currentState = .collapsed
                    self.leftVC = nil
                }
            })
        }
    }
}
```

```
                                                          ↗ default value
func animateCenterPanelXPosition(targetPosition: CGFloat, completion: ((Bool) -> Void)! = nil){
    UIView.animate(withDuration: 0.5, delay: 0, usingSpringWithDamping: 0.8, initialSpringVelocity: 0,
        options: .curveEaseOut, animations: {
        self.centerController.view.frame.origin.x = targetPosition
    }, completion: completion)
}

func setupWhiteCoverView(){
    let whiteCoverView = UIView(frame: CGRect(x:0, y:0, width: view.frame.width,
        height: view.frame.height))
    whiteCoverView.alpha = 0.0
    whiteCoverView.backgroundColor = UIColor.white
    whiteCoverView.tag = 25

    self.centerController.view.addSubview(whiteCoverView)
    whiteCoverView.fadeTo(alphaValue: 0.75, withDuration: 0.2)

    tap = UITapGestureRecognizer(target:self, action: #selector(animateLeftPanel(shouldExpand:)))
    tap.numberOfTapsRequired = 1

    self.centerController.view.addGestureRecognizer(tap)
}




func hideWhiteCoverView(){
    centerController.view.removeGestureRecognizer(tap)

    for subview in self.centerController.view.subviews{
        if subview.tag == 25 {
            UIView.animate(withDuration: 0.2, animations: {
                subView.alpha = 0.0
            }, completion: { (finished) in
                if finished == true{
                    subview.removeFromSuperview()
                }
            })
        }
    }
}

func shouldShowShadowForCenterViewController(_ status: Bool){
    if status {
        centerController.view.layer.shadowOpacity = 0.6
    } else {
        centerController.view.layer.shadowOpacity = 0
    }
}
```

```
func animateStatusBar(){
    UIView.animate(withDuration: 0.5, delay: 0, usingSpringWithDamping: 0.8, initialSpringVelocity: 0,
        options: .curveEaseOut, animations: {
            self.setNeedsStatusBarAppearanceUpdate()
    })
}

private extension UIStoryboard{
    class func mainStoryboard() -> UIStoryboard{
        return UIStoryboard(name: "Main", bundle: Bundle.main)
    }

    class func leftViewController() -> LeftSidePanelVC?{
        return mainStoryboard().instantiateViewController(withIdentifier: "LeftSidePanelVC") as? LeftSidePanelVC
    }
    class func homeVC() -> HomeVC?{
        return storyboard().instantiateViewController(withIdentifier: "HomeVC") as? HomeVC
    }
}
```

// HomeVC.swift (Controller)

```
import UIKit
import MapKit
import RevealingSplashView

class HomeVC: UIViewController, MKMapViewDelegate{
    @IBOutlet weak var mapView: MKMapView!
    @IBOutlet weak var actionButton: RoundedShadowButton!
    var tableView = UITableView()
    var delegate: CenterVCDelegate?
    let revealingSplashView = RevealingSplashView(iconImage: UIImage(named: "launchScreenIcon")!,
        iconInitialSize: CGSize(width: 80, height: 80, backgroundColor: UIColor.white

    override func viewDidLoad(){
        super.viewDidLoad()
        mapView.delegate = self
        destinationTextField.delegate = self
        self.view.addSubview(revealingSplashView)
        revealingSplashView.animationType = SplashAnimationType.heartBeat
        revealingSplashView.startAnimation()

        revealingSplashView.heartAttach = true
    }

    @IBAction func actionBtnWasPressed(_ sender: Any){
        actionBtn.animateButton(shouldLoad: true, withMessage: nil)
    }

    @IBAction func menuBtnWasPressed(_ sender: Any){
        delegate?.toggleLeftPanel()
    }
```

```
import UIKit
import MapKit
import CoreLocation
import RevealingSplashView
import Firebase

enum AnnotationType {
    case pickup
    case destination
    case driver
}

enum ButtonAction {
    case requestRide
    case getDirectionsToPassenger
    case getDirectionsToDestination
    case startTrip
    case endTrip
}

class HomeVC: UIViewController, Alertable {

    @IBOutlet weak var mapView: MKMapView!
    @IBOutlet weak var actionBtn: RoundedShadowButton!
    @IBOutlet weak var centerMapBtn: UIButton!
    @IBOutlet weak var destinationTextField: UITextField!
    @IBOutlet weak var destinationCircle: CircleView!
    @IBOutlet weak var cancelBtn: UIButton!

    var delegate: CenterVCDelegate?

    var manager: CLLocationManager?

    var currentUserId = FIRAuth.auth()?.currentUser?.uid

    var regionRadius: CLLocationDistance = 1000

    let revealingSplashView = RevealingSplashView(iconImage: UIImage(named: "launchScreenIcon")!, iconInitialSize: CGSize(width: 80, height: 80),
backgroundColor: UIColor.white)
    var tableView = UITableView()

    var matchingItems: [MKMapItem] = [MKMapItem]()

    var route: MKRoute?

    var selectedItemPlacemark: MKPlacemark? = nil

    var actionForButton: ButtonAction = .requestRide

    override func viewDidLoad() {
        super.viewDidLoad()

        manager = CLLocationManager()
        manager?.delegate = self
        manager?.desiredAccuracy = kCLLocationAccuracyBest

        checkLocationAuthStatus()

        mapView.delegate = self
        destinationTextField.delegate = self

        centerMapOnUserLocation()

        DataService.instance.REF_DRIVERS.observe(.value, with: { (snapshot) in
            self.loadDriverAnnotationsFromFB()

            DataService.instance.passengerIsOnTrip(passengerKey: self.currentUserId!, handler: { (isOnTrip, driverKey, tripKey) in
                if isOnTrip == true {
                    self.zoom(toFitAnnotationsFromMapView: self.mapView, forActiveTripWithDriver: true, withKey: driverKey)
                }
            })
        })

        cancelBtn.alpha = 0.0

        self.view.addSubview(revealingSplashView)
        revealingSplashView.animationType = SplashAnimationType.heartBeat
        revealingSplashView.startAnimation()

        UpdateService.instance.observeTrips { (tripDict) in
            if let tripDict = tripDict {
                let pickupCoordinateArray = tripDict[USER_PICKUP_COORDINATE] as! NSArray
                let tripKey = tripDict[USER_PASSENGER_KEY] as! String
                let acceptanceStatus = tripDict[TRIP_IS_ACCEPTED] as! Bool
```

```
            if acceptanceStatus == false {
                DataService.instance.driverIsAvailable(key: self.currentUserId!, handler: { (available) in
                    if let available = available {
                        if available == true {
                            let storyboard = UIStoryboard(name: MAIN_STORYBOARD, bundle: Bundle.main)
                            let pickupVC = storyboard.instantiateViewController(withIdentifier: VC_PICKUP) as? PickupVC
                            pickupVC?.initData(coordinate: CLLocationCoordinate2D(latitude: pickupCoordinateArray[0] as! CLLocationDegrees, longitude:
pickupCoordinateArray[1] as! CLLocationDegrees), passengerKey: tripKey)
                            self.present(pickupVC!, animated: true, completion: nil)
                        }
                    }
                })
            }
        }
    }
}
```

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    if currentUserId != nil {
        DataService.instance.userIsDriver(userKey: currentUserId!, handler: { (status) in
            if status == true {
                self.buttonsForDriver(areHidden: true)
            }
        })
    }

    DataService.instance.REF_TRIPS.observe(.childRemoved, with: { (removedTripSnapshot) in
        let removedTripDict = removedTripSnapshot.value as? [String: AnyObject]
        if removedTripDict?[DRIVER_KEY] != nil {
            DataService.instance.REF_DRIVERS.child(removedTripDict?[DRIVER_KEY] as! String).updateChildValues([DRIVER_IS_ON_TRIP: false])
        }

        DataService.instance.userIsDriver(userKey: self.currentUserId!, handler: { (isDriver) in
            if isDriver == true {
                self.removeOverlaysAndAnnotations(forDrivers: false, forPassengers: true)
            } else {
                self.cancelBtn.fadeTo(alphaValue: 0.0, withDuration: 0.2)
                self.actionBtn.animateButton(shouldLoad: false, withMessage: MSG_REQUEST_RIDE)

                self.destinationTextField.isUserInteractionEnabled = true
                self.destinationTextField.text = ""

                self.removeOverlaysAndAnnotations(forDrivers: false, forPassengers: true)
                self.centerMapOnUserLocation()
            }
        })
    })

    if currentUserId != nil {
        DataService.instance.driverIsOnTrip(driverKey: self.currentUserId!, handler: { (isOnTrip, driverKey, tripKey) in
            if isOnTrip == true {
                DataService.instance.REF_TRIPS.observeSingleEvent(of: .value, with: { (tripSnapshot) in
                    if let tripSnapshot = tripSnapshot.children.allObjects as? [FIRDataSnapshot] {
                        for trip in tripSnapshot {
                            if trip.childSnapshot(forPath: DRIVER_KEY).value as? String == self.currentUserId! {
                                let pickupCoordinatesArray = trip.childSnapshot(forPath: USER_PICKUP_COORDINATE).value as! NSArray
                                let pickupCoordinate: CLLocationCoordinate2D = CLLocationCoordinate2D(latitude: pickupCoordinatesArray[0] as!
CLLocationDegrees, longitude: pickupCoordinatesArray[1] as! CLLocationDegrees)
                                let pickupPlacemark = MKPlacemark(coordinate: pickupCoordinate)

                                self.dropPinFor(placemark: pickupPlacemark)
                                self.searchMapKitForResultsWithPolyline(forOriginMapItem: nil, withDestinationMapItem: MKMapItem(placemark: pickupPlacemark))

                                self.setCustomRegion(forAnnotationType: .pickup, withCoordinate: pickupCoordinate)

                                self.actionForButton = .getDirectionsToPassenger
                                self.actionBtn.setTitle(MSG_GET_DIRECTIONS, for: .normal)

                                self.buttonsForDriver(areHidden: false)
                            }
                        }
                    }
                })
            }
        })
        connectUserAndDriverForTrip()
    }
}
```

```
func mapView(_ mapView: MKMapView, regionWillChangeAnimated animated: Bool) {
    centerMapBtn.fadeTo(alphaValue: 1.0, withDuration: 0.2)
}

func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer {
    let lineRenderer = MKPolylineRenderer(overlay: (self.route?.polyline)!)
    lineRenderer.strokeColor = UIColor(red: 216/255, green: 71/255, blue: 30/255, alpha: 0.75)
    lineRenderer.lineWidth = 3

    shouldPresentLoadingView(false)

    return lineRenderer
}

func performSearch() {
    matchingItems.removeAll()
    let request = MKLocalSearchRequest()
    request.naturalLanguageQuery = destinationTextField.text
    request.region = mapView.region

    let search = MKLocalSearch(request: request)

    search.start { (response, error) in
        if error != nil {
            self.showAlert(ERROR_MSG_UNEXPECTED_ERROR)
        } else if response!.mapItems.count == 0 {
            self.showAlert(ERROR_MSG_NO_MATCHES_FOUND)
        } else {
            for mapItem in response!.mapItems {
                self.matchingItems.append(mapItem as MKMapItem)
                self.tableView.reloadData()
                self.shouldPresentLoadingView(false)
            }
        }
    }
}

func dropPinFor(placemark: MKPlacemark) {
    selectedItemPlacemark = placemark

    for annotation in mapView.annotations {
        if annotation.isKind(of: MKPointAnnotation.self) {
            mapView.removeAnnotation(annotation)
        }
    }

    let annotation = MKPointAnnotation()
    annotation.coordinate = placemark.coordinate
    mapView.addAnnotation(annotation)
}

func searchMapKitForResultsWithPolyline(forOriginMapItem originMapItem: MKMapItem?, withDestinationMapItem destinationMapItem: MKMapItem) {
    let request = MKDirectionsRequest()

    if originMapItem == nil {
        request.source = MKMapItem.forCurrentLocation()
    } else {
        request.source = originMapItem
    }

    request.destination = destinationMapItem
    request.transportType = MKDirectionsTransportType.automobile
    request.requestsAlternateRoutes = true

    let directions = MKDirections(request: request)

    directions.calculate { (response, error) in
        guard let response = response else {
            self.showAlert(error.debugDescription)
            return
        }
        self.route = response.routes[0]

        self.mapView.add(self.route!.polyline)

        self.zoom(toFitAnnotationsFromMapView: self.mapView, forActiveTripWithDriver: false, withKey: nil)

        let delegate = AppDelegate.getAppDelegate()
        delegate.window?.rootViewController?.shouldPresentLoadingView(false)
    }
}
```

```
    func zoom(toFitAnnotationsFromMapView mapView: MKMapView, forActiveTripWithDriver: Bool, withKey key: String?) {
        if mapView.annotations.count == 0 {
            return
        }

        var topLeftCoordinate = CLLocationCoordinate2D(latitude: -90, longitude: 180)
        var bottomRightCoordinate = CLLocationCoordinate2D(latitude: 90, longitude: -180)


        if forActiveTripWithDriver {
            for annotation in mapView.annotations {
                if let annotation = annotation as? DriverAnnotation {
                    if annotation.key == key {
                        topLeftCoordinate.longitude = fmin(topLeftCoordinate.longitude, annotation.coordinate.longitude)
                        topLeftCoordinate.latitude = fmax(topLeftCoordinate.latitude, annotation.coordinate.latitude)
                        bottomRightCoordinate.longitude = fmax(bottomRightCoordinate.longitude, annotation.coordinate.longitude)
                        bottomRightCoordinate.latitude = fmin(bottomRightCoordinate.latitude, annotation.coordinate.latitude)
                    }
                } else {
                    topLeftCoordinate.longitude = fmin(topLeftCoordinate.longitude, annotation.coordinate.longitude)
                    topLeftCoordinate.latitude = fmax(topLeftCoordinate.latitude, annotation.coordinate.latitude)
                    bottomRightCoordinate.longitude = fmax(bottomRightCoordinate.longitude, annotation.coordinate.longitude)
                    bottomRightCoordinate.latitude = fmin(bottomRightCoordinate.latitude, annotation.coordinate.latitude)
                }
            }
        }


        for annotation in mapView.annotations where !annotation.isKind(of: DriverAnnotation.self) {
            topLeftCoordinate.longitude = fmin(topLeftCoordinate.longitude, annotation.coordinate.longitude)
            topLeftCoordinate.latitude = fmax(topLeftCoordinate.latitude, annotation.coordinate.latitude)
            bottomRightCoordinate.longitude = fmax(bottomRightCoordinate.longitude, annotation.coordinate.longitude)
            bottomRightCoordinate.latitude = fmin(bottomRightCoordinate.latitude, annotation.coordinate.latitude)
        }

        var region = MKCoordinateRegion(center: CLLocationCoordinate2DMake(topLeftCoordinate.latitude - (topLeftCoordinate.latitude -
bottomRightCoordinate.latitude) * 0.5, topLeftCoordinate.longitude + (bottomRightCoordinate.longitude - topLeftCoordinate.longitude) * 0.5), span:
MKCoordinateSpan(latitudeDelta: fabs(topLeftCoordinate.latitude - bottomRightCoordinate.latitude) * 2.0, longitudeDelta:
fabs(bottomRightCoordinate.longitude - topLeftCoordinate.longitude) * 2.0))

        region = mapView.regionThatFits(region)
        mapView.setRegion(region, animated: true)
    }

    func removeOverlaysAndAnnotations(forDrivers: Bool?, forPassengers: Bool?) {

        for annotation in mapView.annotations {
            if let annotation = annotation as? MKPointAnnotation {
                mapView.removeAnnotation(annotation)
            }

            if forPassengers! {
                if let annotation = annotation as? PassengerAnnotation {
                    mapView.removeAnnotation(annotation)
                }
            }

            if forDrivers! {
                if let annotation = annotation as? DriverAnnotation {
                    mapView.removeAnnotation(annotation)
                }
            }
        }

        for overlay in mapView.overlays {
            if overlay is MKPolyline {
                mapView.remove(overlay)
            }
        }
    }

    func setCustomRegion(forAnnotationType type: AnnotationType, withCoordinate coordinate: CLLocationCoordinate2D) {
        if type == .pickup {
            let pickupRegion = CLCircularRegion(center: coordinate, radius: 100, identifier: REGION_PICKUP)
            manager?.startMonitoring(for: pickupRegion)
        } else if type == .destination {
            let destinationRegion = CLCircularRegion(center: coordinate, radius: 100, identifier: REGION_DESTINATION)
            manager?.startMonitoring(for: destinationRegion)
        }
    }
}
```

```
extension HomeVC: UITextFieldDelegate {
    func textFieldDidBeginEditing(_ textField: UITextField) {
        if textField == destinationTextField {
            tableView.frame = CGRect(x: 20, y: view.frame.height, width: view.frame.width - 40, height: view.frame.height - 170)
            tableView.layer.cornerRadius = 5.0
            tableView.register(UITableViewCell.self, forCellReuseIdentifier: CELL_LOCATION)

            tableView.delegate = self
            tableView.dataSource = self

            tableView.tag = 18
            tableView.rowHeight = 60

            view.addSubview(tableView)
            animateTableView(shouldShow: true)

            UIView.animate(withDuration: 0.2, animations: {
                self.destinationCircle.backgroundColor = UIColor.red
                self.destinationCircle.borderColor = UIColor.init(red: 199/255, green: 0/255, blue: 0/255, alpha: 1.0)
            })
        }
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        if textField == destinationTextField {
            performSearch()
            shouldPresentLoadingView(true)
            view.endEditing(true)
        }
        return true
    }

    func textFieldDidEndEditing(_ textField: UITextField) {
        if textField == destinationTextField {
            if destinationTextField.text == "" {
                UIView.animate(withDuration: 0.2, animations: {
                    self.destinationCircle.backgroundColor = UIColor.lightGray
                    self.destinationCircle.borderColor = UIColor.darkGray
                })
            }
        }
    }

    func textFieldShouldClear(_ textField: UITextField) -> Bool {
        matchingItems = []
        tableView.reloadData()

        DataService.instance.REF_USERS.child(currentUserId!).child(TRIP_COORDINATE).removeValue()

        mapView.removeOverlays(mapView.overlays)
        for annotation in mapView.annotations {
            if let annotation = annotation as? MKPointAnnotation {
                mapView.removeAnnotation(annotation)
            } else if annotation.isKind(of: PassengerAnnotation.self) {
                mapView.removeAnnotation(annotation)
            }
        }

        centerMapOnUserLocation()
        return true
    }

    func animateTableView(shouldShow: Bool) {
        if shouldShow {
            UIView.animate(withDuration: 0.2, animations: {
                self.tableView.frame = CGRect(x: 20, y: 170, width: self.view.frame.width - 40, height: self.view.frame.height - 170)
            })
        } else {
            UIView.animate(withDuration: 0.2, animations: {
                self.tableView.frame = CGRect(x: 20, y: self.view.frame.height, width: self.view.frame.width - 40, height: self.view.frame.height - 170)
            }, completion: { (finished) in
                for subview in self.view.subviews {
                    if subview.tag == 18 {
                        subview.removeFromSuperview()
                    }
                }
            })
        }
    }
}
```

(set storyboard ⇒ clear button ⇒ show while editing

```
extension HomeVC: UITableViewDelegate, UITableViewDataSource {
    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = UITableViewCell(style: .subtitle, reuseIdentifier: CELL_LOCATION)
        let mapItem = matchingItems[indexPath.row]
        cell.textLabel?.text = mapItem.name
        cell.detailTextLabel?.text = mapItem.placemark.title
        return cell
    }

    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return matchingItems.count
    }

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

        shouldPresentLoadingView(true)

        let passengerCoordinate = manager?.location?.coordinate

        let passengerAnnotation = PassengerAnnotation(coordinate: passengerCoordinate!, key: currentUserId!)
        mapView.addAnnotation(passengerAnnotation)

        destinationTextField.text = tableView.cellForRow(at: indexPath)?.textLabel?.text

        let selectedMapItem = matchingItems[indexPath.row]

        DataService.instance.REF_USERS.child(currentUserId!).updateChildValues([TRIP_COORDINATE: [selectedMapItem.placemark.coordinate.latitude,
selectedMapItem.placemark.coordinate.longitude]])

        dropPinFor(placemark: selectedMapItem.placemark)

        searchMapKitForResultsWithPolyline(forOriginMapItem: nil, withDestinationMapItem: selectedMapItem)

        animateTableView(shouldShow: false)
    }

    func scrollViewDidScroll(_ scrollView: UIScrollView) {
        view.endEditing(true)
    }

    func scrollViewWillBeginDragging(_ scrollView: UIScrollView) {
        if destinationTextField.text == "" {
            animateTableView(shouldShow: false)
        }
    }
}
```

// LeftSidePanelVC.swift (controller)

```
import UIKit
import Firebase

class LeftSidePanelVC: UIViewController {

    let appDelegate = AppDelegate.getAppDelegate()

    let currentUserId = FIRAuth.auth()?.currentUser?.uid

    @IBOutlet weak var userEmailLbl: UILabel!
    @IBOutlet weak var userAccountTypeLbl: UILabel!
    @IBOutlet weak var userImageView: RoundImageView!
    @IBOutlet weak var loginOutBtn: UIButton!
    @IBOutlet weak var pickupModeSwitch: UISwitch!
    @IBOutlet weak var pickupModeLbl: UILabel!

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

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        pickupModeSwitch.isOn = false
        pickupModeSwitch.isHidden = true
        pickupModeLbl.isHidden = true

        observePassengersAndDrivers()

        if FIRAuth.auth()?.currentUser == nil {
            userEmailLbl.text = ""
            userAccountTypeLbl.text = ""
            userImageView.isHidden = true
            loginOutBtn.setTitle(MSG_SIGN_UP_SIGN_IN, for: .normal)
        } else {
            userEmailLbl.text = FIRAuth.auth()?.currentUser?.email
            userAccountTypeLbl.text = ""
            userImageView.isHidden = false
            loginOutBtn.setTitle(MSG_SIGN_OUT, for: .normal)
        }
    }

    func observePassengersAndDrivers() {
        DataService.instance.REF_USERS.observeSingleEvent(of: .value, with: { (snapshot) in
            if let snapshot = snapshot.children.allObjects as? [FIRDataSnapshot] {
                for snap in snapshot {
                    if snap.key == FIRAuth.auth()?.currentUser?.uid {
                        self.userAccountTypeLbl.text = ACCOUNT_TYPE_PASSENGER
                    }
                }
            }
        })

        DataService.instance.REF_DRIVERS.observeSingleEvent(of: .value, with: { (snapshot) in
            if let snapshot = snapshot.children.allObjects as? [FIRDataSnapshot] {
                for snap in snapshot {
                    if snap.key == FIRAuth.auth()?.currentUser?.uid {
                        self.userAccountTypeLbl.text = ACCOUNT_TYPE_DRIVER
                        self.pickupModeSwitch.isHidden = false

                        let switchStatus = snap.childSnapshot(forPath: ACCOUNT_PICKUP_MODE_ENABLED).value as! Bool
                        self.pickupModeSwitch.isOn = switchStatus
                        self.pickupModeLbl.isHidden = false
                    }
                }
            }
        })
    }

    @IBAction func switchWasToggled(_ sender: Any) {
        if pickupModeSwitch.isOn {
            pickupModeLbl.text = MSG_PICKUP_MODE_ENABLED
            appDelegate.MenuContainerVC.toggleLeftPanel()
            DataService.instance.REF_DRIVERS.child(currentUserId!).updateChildValues([ACCOUNT_PICKUP_MODE_ENABLED: true])
        } else {
            pickupModeLbl.text = MSG_PICKUP_MODE_DISABLED
            appDelegate.MenuContainerVC.toggleLeftPanel()
            DataService.instance.REF_DRIVERS.child(currentUserId!).updateChildValues([ACCOUNT_PICKUP_MODE_ENABLED: false])
        }
    }
```

```
    @IBAction func signUpLoginBtnWasPressed(_ sender: Any) {
        if FIRAuth.auth()?.currentUser == nil {
            let storyboard = UIStoryboard(name: MAIN_STORYBOARD, bundle: Bundle.main)
            let loginVC = storyboard.instantiateViewController(withIdentifier: VC_LOGIN) as? LoginVC
            present(loginVC!, animated: true, completion: nil)
        } else {
            do {
                try FIRAuth.auth()?.signOut()
                userEmailLbl.text = ""
                userAccountTypeLbl.text = ""
                userImageView.isHidden = true
                pickupModeLbl.text = ""
                pickupModeSwitch.isHidden = true
                loginOutBtn.setTitle(MSG_SIGN_UP_SIGN_IN, for: .normal)
            } catch (let error) {
                print(error)
            }
        }
    }
}
```

```
// LoginVC.swift          (Controller)
import UIKit
import Firebase
class loginVC: UIViewController, UITextFieldDelegate {

    @IBOutlet weak var emailField: RoundedCornerTextField!
    @IBOutlet weak var passwordField: RoundedCornerTextField!
    @IBOutlet weak var segmentedControl: UISegmentedControl!
    @IBOutlet weak var authBtn: RoundedShadowButton!

    override func viewDidLoad(){
        super.viewDidLoad()
        emailField.delegate = self
        passwordField.delegate = self
        view.bindToKeyboard()

        let tap = UITapGestureRecognizer(target: self, action: #selector(handleScreenTap(sender:)))
        self.view.addGestureRecognizer(tap)

    }
    func handleScreenTap(sender: UITapGestureRecognizer){
        self.view.endEditing(true)
    }

    @IBAction func cancelBtnWasPressed(_ sender: Any){
        dismiss(animated: true, completion: nil)
    }

    @IBAction func authBtnWasPressed(_ sender: Any){
        if emailField.text != nil && passwordField.text != nil{
            authBtn.animateButton(shouldLoad: true, withMessage: nil)
            self.view.endEditing(true)
            if let email = emailField.text, let password = passwordField.text{
                FireAuth.auth()?.signIn(withEmail: email, password: password,
                    completion: {
                    if error == nil{
                        if let user == user{
                            if self.segmentedControl.selectedSegmentIndex == 0{
                                let userData = ["provider": user.providerID] as [String: Any]
                                DataService.instance.createFirebaseDBUser(uid: user.id, userData: userData, isDriver: false)
                            }else{
                                let userData = ["provider": user.providerID, "userIsDriver": true, "isPickUpModeEnabled": false,
                                    "driverIsOnTrip": false] as [String: Any]
                                DataService.instance.createFirebaseDBUser(uid: user.uid, userData: userData, isDriver: true)
                            }
                        }
                        print("success login")
                        self.dismiss(animated: true, completion: nil)
                    }else{
                        if let errorCode = FIRAuthErrorCode(rawValue: error!.__code){   // must have __
                            switch errorCode{
                                case .errorCodeWrongPassword:
                                    print...
```

```
default:
    print ....
}
}

FIRAuth.auth()?.createUser(withEmail: email, password: password, completion: {
    (user, error) in
    if error != nil{                                          must have _
        if let errorCode = FIRAuthErrorCode(rawValue: error!._code){
            switch errorCode{
                case .errorCodeInvalidEmail:
                    print ...
                default:
                    print ....
            }
        }
    } else{
        if let user = user{
            if self.segmentedControl.selectedSegmentIndex == 0{
                let userData = ["provider": user.providerID] as [String:Any]
                DataService.instance.createFirebaseDBUser(uid: user.id, userData: userData, isDriver: false)
            } else{
                let userData = ["provider": user.providerID, "userIsDriver": true, "isPickUpModeEnabled": false,
                    "driverIsOnTrip": false] as [String: Any]
                DataService.instance.createFirebaseDBUser(uid: user.uid, userData: userData, isDriver: true)
            }
        }
        print("success create")    // add callbacks to the create user func in real project
        self.dismiss(animated: true, completion: nil)

    }
})
}
}
}
}
}
```