



PROF. RENÊ XAVIER

DESENVOLVIMENTO PARA IOS 11 COM SWIFT 4

ONDE ENCONTRAR O MATERIAL?

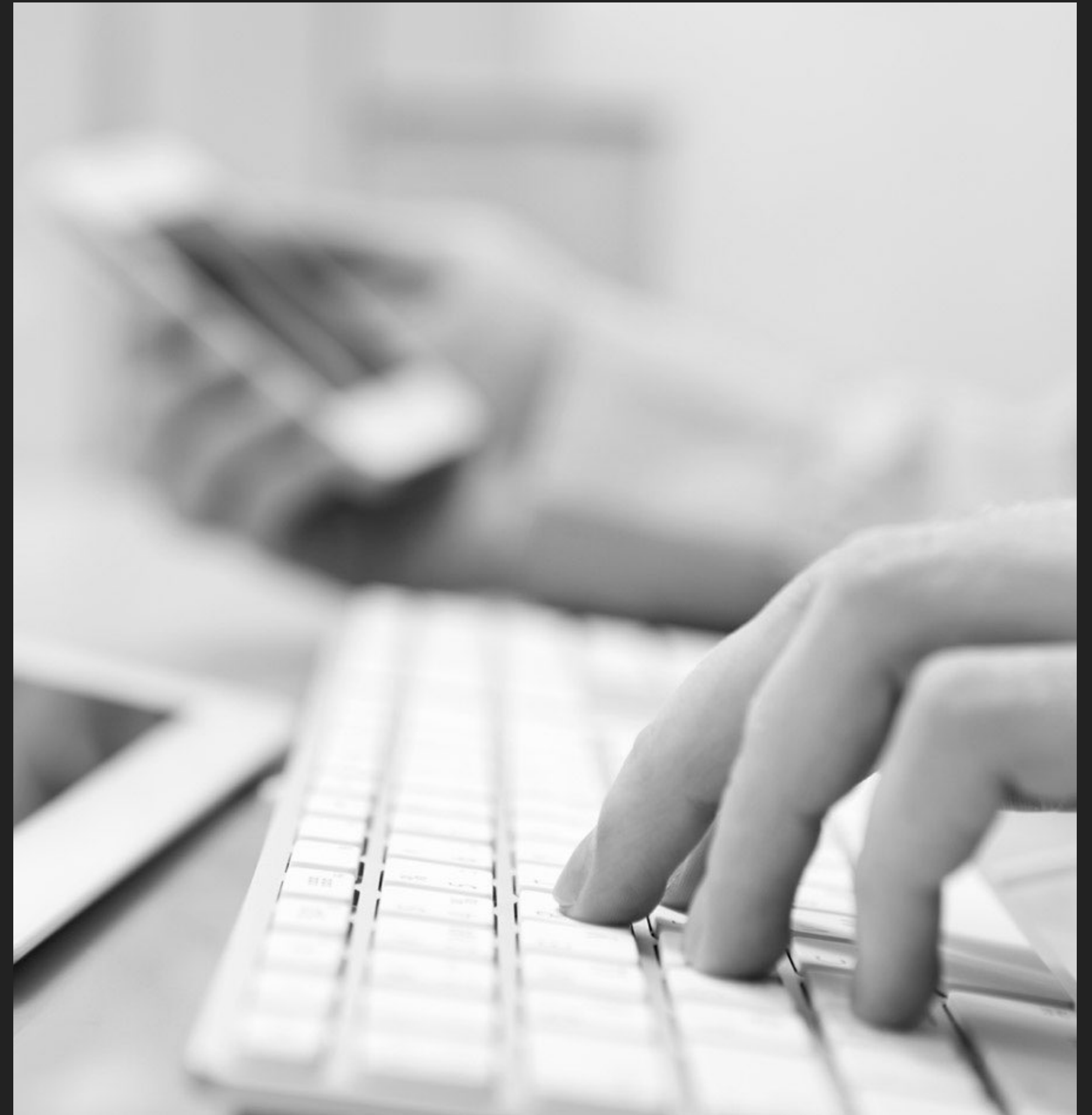
[HTTPS://GITHUB.COM/RENEFX/IOS-2018-01](https://github.com/RENEFX/IOS-2018-01)

GITHUB

ALÉM DO TRADICIONAL BLACKBOARD DO IESB

AGENDA

- ▶ Localização
- ▶ MapKit
- ▶ Migração do Realm





LOCALIZAÇÃO

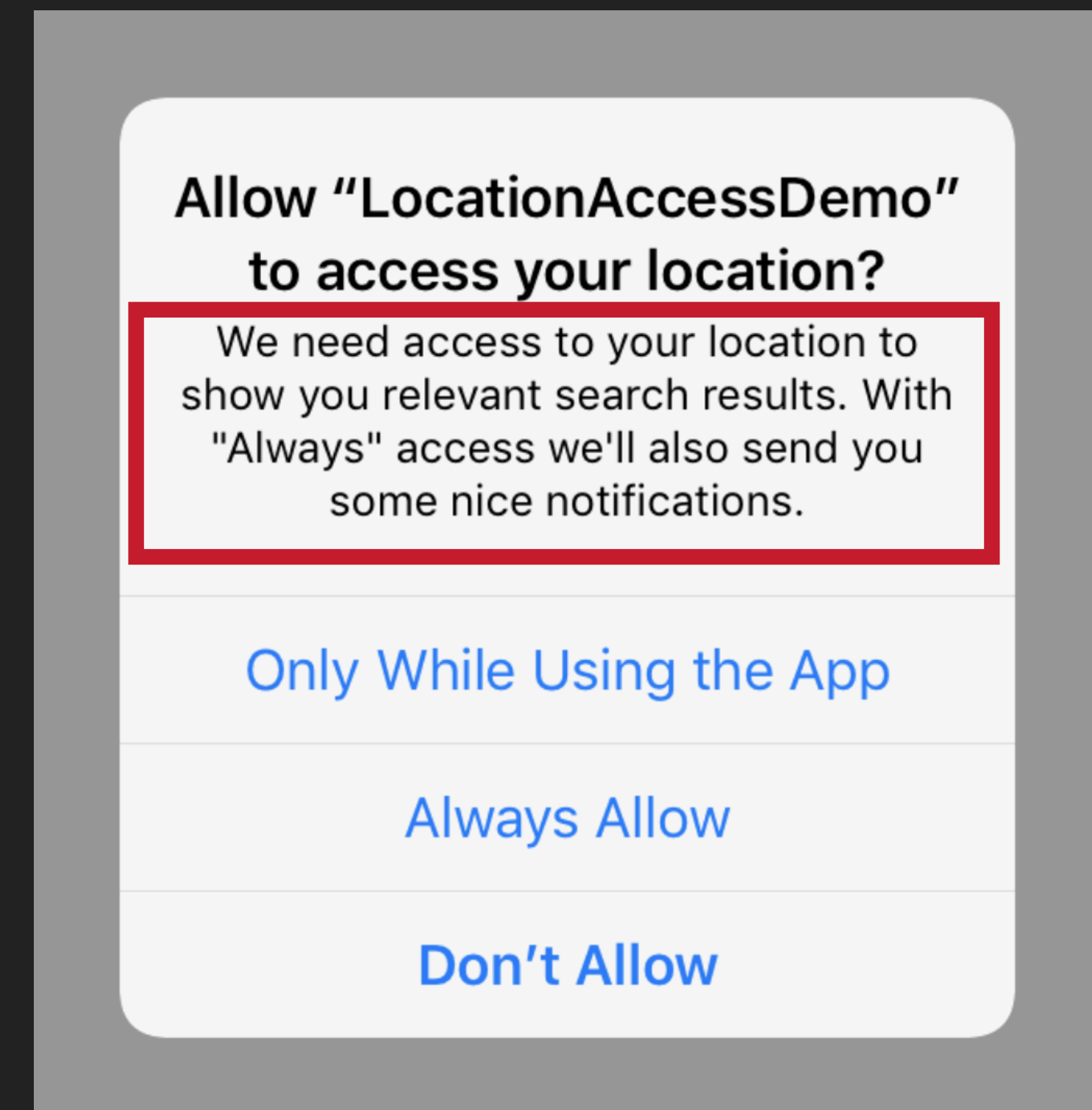
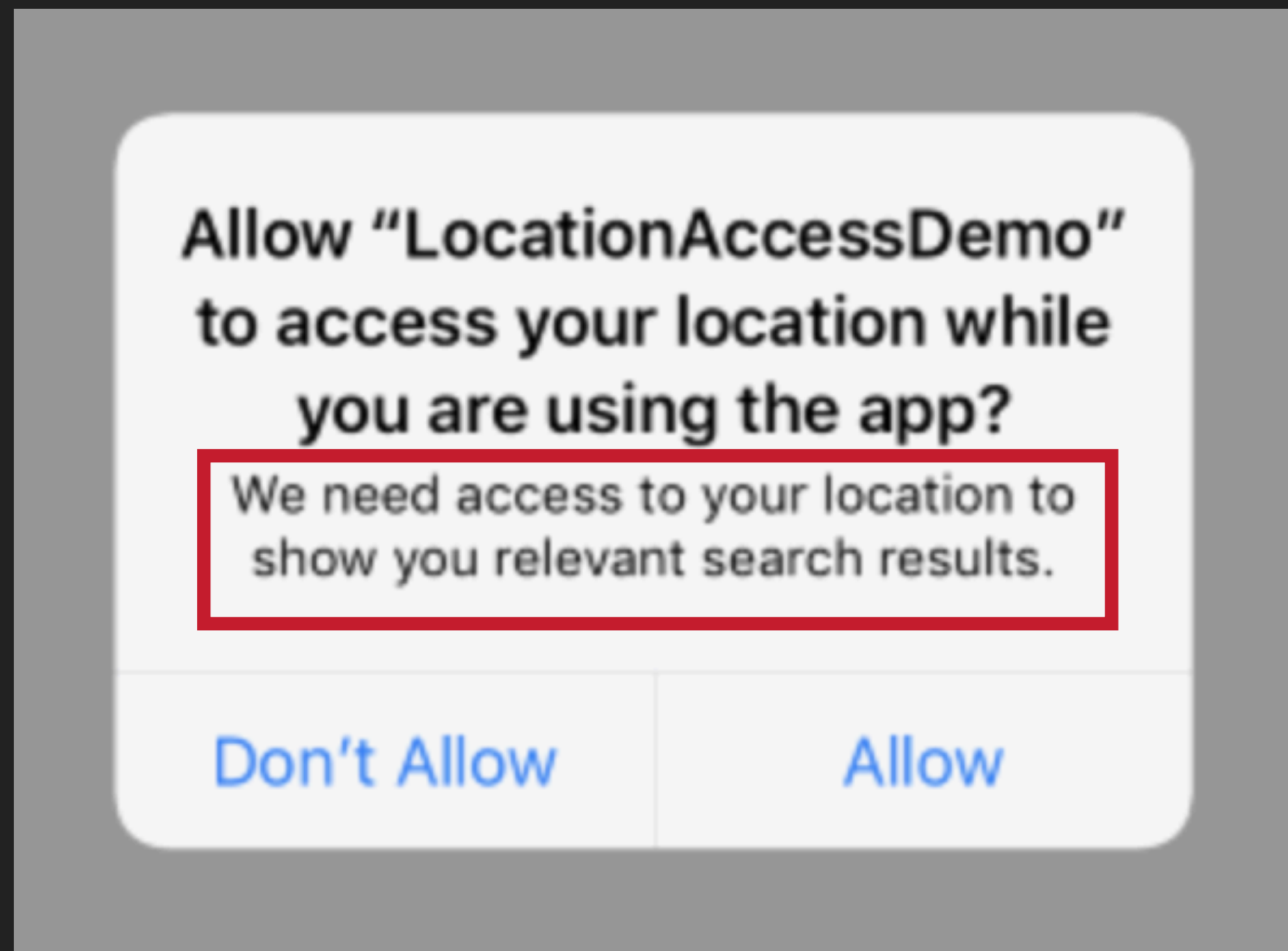
₃

LOCALIZAÇÃO

Para determinar a localização do usuário não precisamos de exibir um mapa

Podemos coletar as informações sem o usuário ver

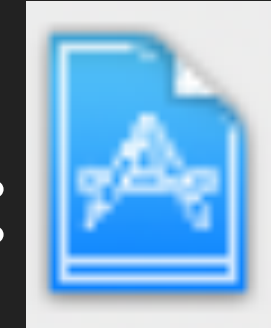
Mas não sem antes pedir a permissão dele



LOCALIZAÇÃO

Antes de falar de localização, temos que falar sobre o arquivo info.plist:

- ▶ Plist - Property list
- ▶ Possui informações referentes ao projeto:



- ▶ Algumas repetidas do nosso projeto:

- ▶ Número da versão
 - ▶ Número do Build
 - ▶ Nome da Storyboard inicial
 - ▶ Nome da nossa Launch Screen



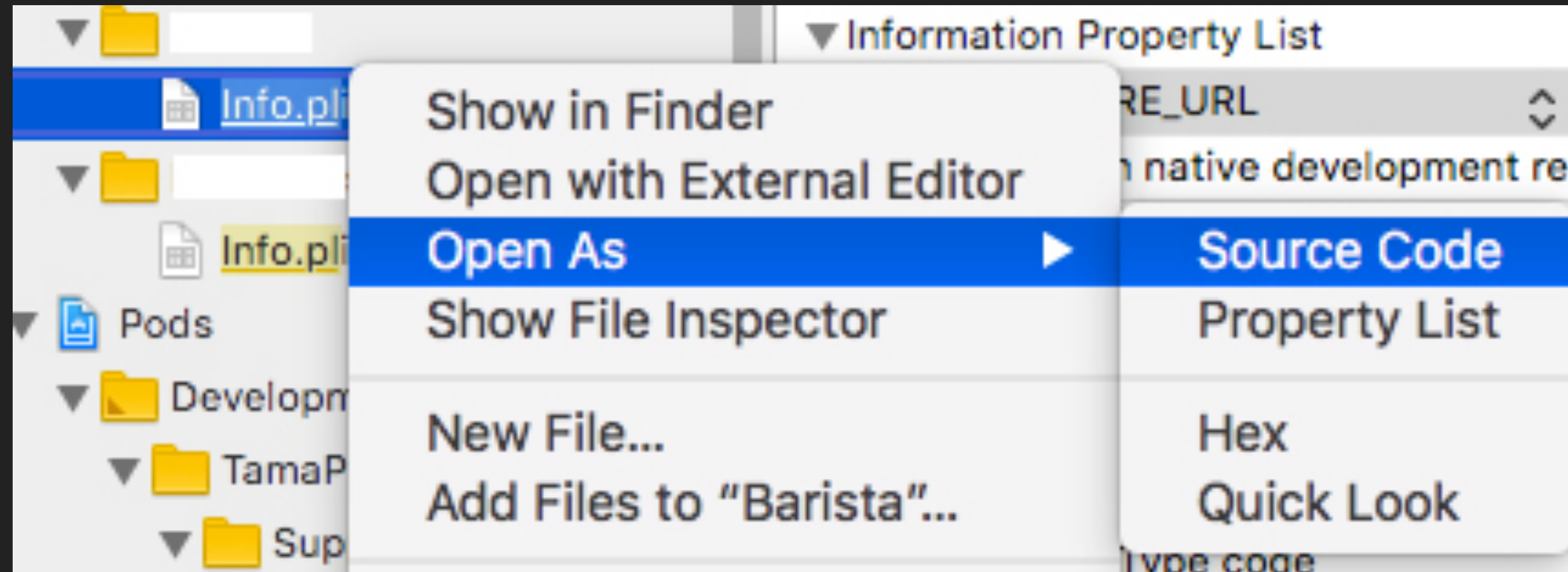
Info.plist

LOCALIZAÇÃO

Desde o iOS8, temos que pedir a permissão para usar a localização do usuário:

- ▶ O texto a ser exibido é definido no nosso info.plist
- ▶ Caso não seja definida essa descrição, o serviço de localização não será atualizado
- ▶ Podemos pedir dois tipos de acesso a localização:
 - ▶ Quando em uso - `CLLocationWhenInUseUsageDescription`
 - ▶ Sempre
 - ▶ `CLLocationAlwaysAndWhenInUseUsageDescription`
 - ▶ `CLLocationAlwaysUsageDescription` - Se o seu app suporta iOS10 você deve usar esse e o `WhenInUse` além do anterior

LOCALIZAÇÃO



```
<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
<string>Usamos os seus dados para traçar rotas sempre</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>Usamos os seus dados para traçar rotas em background</string>
<key>NSLocationWhenInUseUsageDescription</key>
<string>Usamos os seus dados para traçar rotas somente quando você estiver usando o App</string>
```


LOCALIZAÇÃO

QUEM CUIDA D
LOCALIZAÇÃO É O
CLLOCATIONMANAGER

LOCALIZAÇÃO

PARA PEGAR A
LOCALIZAÇÃO
UTILIZAMOS
O QUE?

DELEGATE

LOCALIZAÇÃO

Vamos então:

- Importar CoreLocation
- Declarar na classe que implementamos o protocolo
- Criar uma instância do CLLocationManager
- Dizer que implementamos o delegate dele
- **Requisitar a permissão do usuário**
- **Iniciar a coleta da localização**
- Implementar o método que é chamado após a localização ser encontrada

LOCALIZAÇÃO

Importar CoreLocation

Declarar na classe que implementamos o protocolo

```
import UIKit
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {
```

LOCALIZAÇÃO

Importar CoreLocation

Declarar na classe que implementamos o protocolo

Criar uma instância do CLLocationManager

```
import UIKit
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

    var locationManager = CLLocationManager()
```

LOCALIZAÇÃO

Dizer que implementamos o delegate

```
import UIKit
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

    var locationManager = CLLocationManager()

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

    func preparaLocalizacao() {
        locationManager.delegate = self
    }
}
```

LOCALIZAÇÃO

Requisitar a permissão do usuário

```
locationManager.requestAlwaysAuthorization()
```

ou

```
locationManager.requestWhenInUseAuthorization()
```


LOCALIZAÇÃO

Requisitar a permissão do usuário

Iniciar a coleta da localização

```
import UIKit
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

    var locationManager = CLLocationManager()

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

    func preparaLocalizacao() {
        locationManager.delegate = self
        locationManager.requestWhenInUseAuthorization()

        locationManager.startUpdatingLocation()
    }
}
```

LOCALIZAÇÃO

Implementar o método que é chamado após a localização ser encontrada

didUpdateLocation

```
import UIKit
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

    var locationManager = CLLocationManager()

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

    func preparaLocalizacao() {
        locationManager.delegate = self
        locationManager.requestWhenInUseAuthorization()

        locationManager.startUpdatingLocation()
    }

    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
        let userLocation :CLLocation = locations[0] as CLLocation
    }
}
```

LOCALIZAÇÃO

Implementar o método que é chamado após a localização ser encontrada

didUpdateLocation

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {  
    let userLocation :CLLocation = locations[0] as CLLocation  
  
    print(userLocation.coordinate.latitude)  
    print(userLocation.coordinate.longitude)  
  
}
```

LOCALIZAÇÃO

Desde o iOS8, temos que pedir a permissão para usar a localização do usuário:

- ▶ Se o usuário negar uma vez, o alerta não será mais exibido

Dessa forma coletamos a localização

Como fazemos para parar?

LOCALIZAÇÃO

Se você quiser atualizar enquanto ele estiver na tela, só pare quando sair:

```
override func viewWillDisappear(_ animated: Bool) {  
    locationManager.stopUpdatingLocation()  
}
```

Se você quiser só uma vez, coloque assim que o método for chamado retornando a localização:


```
func locationManager(_ manager: CLLocationManager,  
                     didUpdateLocations locations: [CLLocation]) {  
    locationManager.stopUpdatingLocation()  
}
```

LOCALIZAÇÃO

Podemos determinar a precisão da localização

O filtro de distância são quantos metros o usuário irá se mover até você ter uma notificação

```
locationManager.delegate = self  
locationManager.desiredAccuracy = kCLLocationAccuracyBest  
locationManager.distanceFilter = 100  
locationManager.requestWhenInUseAuthorization()
```



LOCALIZAÇÃO

Podemos saber quando a permissão mudou

```
func locationManager(_ manager: CLLocationManager, didChangeAuthorization status: CLAuthorizationStatus) {  
    switch status {  
    case .authorizedAlways:  
        break  
    case .authorizedWhenInUse:  
        break  
    case .denied:  
        break  
    case .notDetermined:  
        break  
    case .restricted:  
        break  
    }  
}
```

Esse método será chamado mesmo se a pessoa sair e voltar ao seu app

LOCALIZAÇÃO

Com o CLGeocoder conseguimos encontrar informações dessa localização:

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    let userLocation :CLLocation = locations[0] as CLLocation

    print(userLocation.coordinate.latitude)
    print(userLocation.coordinate.longitude)
    locationManager.stopUpdatingLocation()

    let geocoder = CLGeocoder()
    geocoder.reverseGeocodeLocation(userLocation) { (placemarks, error) in
        if let placemarks = placemarks, placemarks.count > 0 {
            let placemark = placemarks[0]
            print(placemark.country!)
            print(placemark.location?.floor ?? "")
        }
    }
}
```



```
// address dictionary properties
open var name: String? { get } // eg. Apple Inc.

open var thoroughfare: String? { get } // street name, eg. Infinite Loop

open var subThoroughfare: String? { get } // eg. 1

open var locality: String? { get } // city, eg. Cupertino

open var subLocality: String? { get } // neighborhood, common name, eg. Mission District

open var administrativeArea: String? { get } // state, eg. CA

open var subAdministrativeArea: String? { get } // county, eg. Santa Clara

open var postalCode: String? { get } // zip code, eg. 95014

open var isoCountryCode: String? { get } // eg. US

open var country: String? { get } // eg. United States

open var inlandWater: String? { get } // eg. Lake Tahoe

open var ocean: String? { get } // eg. Pacific Ocean

open var areasOfInterest: [String]? { get } // eg. Golden Gate Park
```



MAPKIT

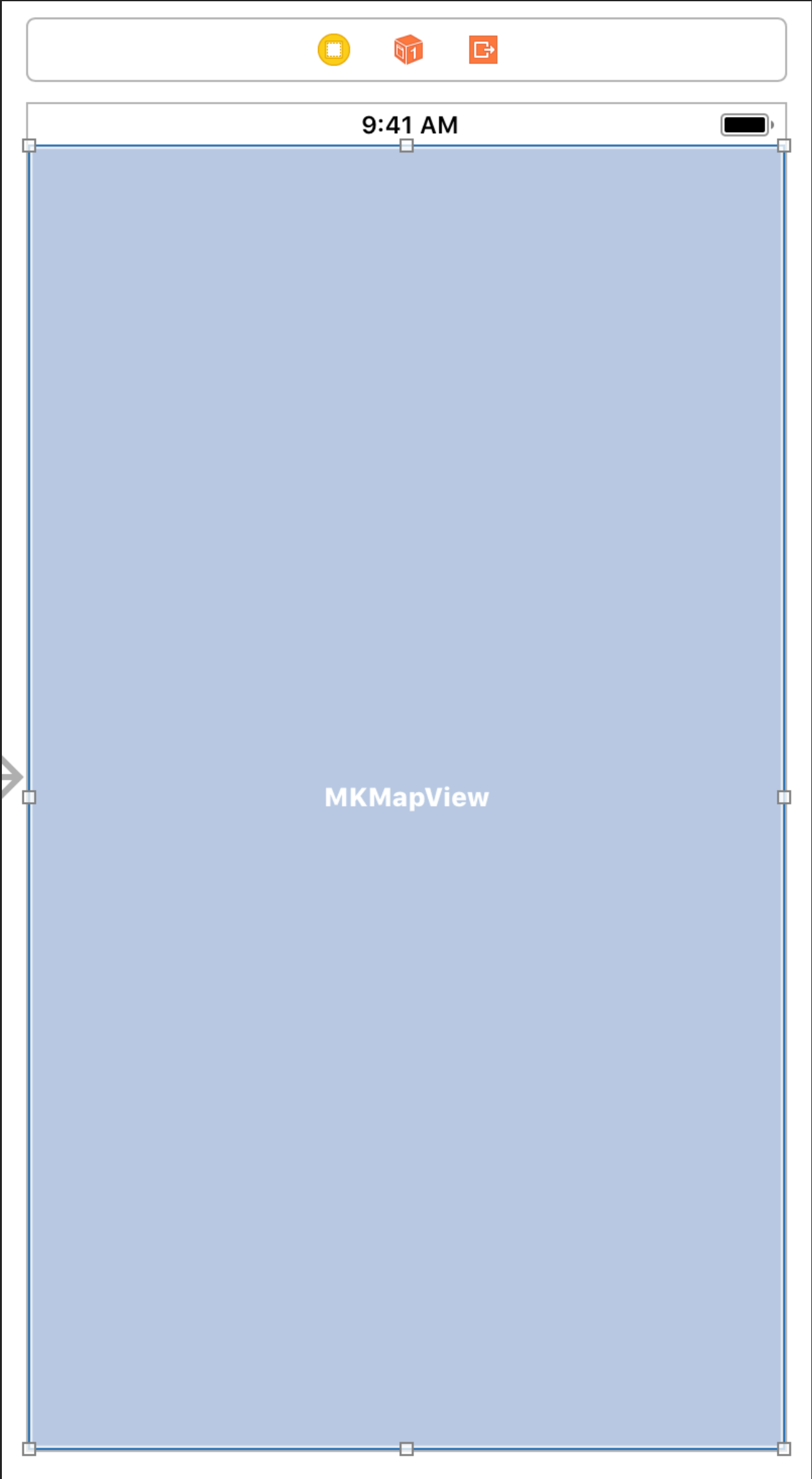
Não é necessário ter pego a localização do usuário para criar um mapa

Só precisamos do MapKit na nossa UIViewController do Storyboard

Podemos adicionar como uma View normal e colocar as constraints nele



Map Kit View - Displays maps and provides an embeddable interface to navigate map content.



MAPKIT

Para customizar, temos que importar o MapKit na nossa ViewController

Vamos também fazer uma outlet do nosso mapa

```
import MapKit
```

```
@IBOutlet weak var mapView: MKMapView!
```

MAPKIT

Para exibir a localização atual do usuário, não precisamos estar atualizando a localização a todo momento
(startUpdatingLocation())

Mas precisamos pedir a autorização do usuário

```
locationManager.requestWhenInUseAuthorization()  
self.mapView.showsUserLocation = true
```


É possível selecionar o tipo do mapa

```
self.mapView.mapType = .|
```

MKMapType hybrid

MKMapType hybridFlyover

MKMapType mutedStandard

MKMapType satellite

MKMapType satelliteFlyover

MKMapType standard

A satellite image of the area with road and road name information layered on top.

MAPKIT

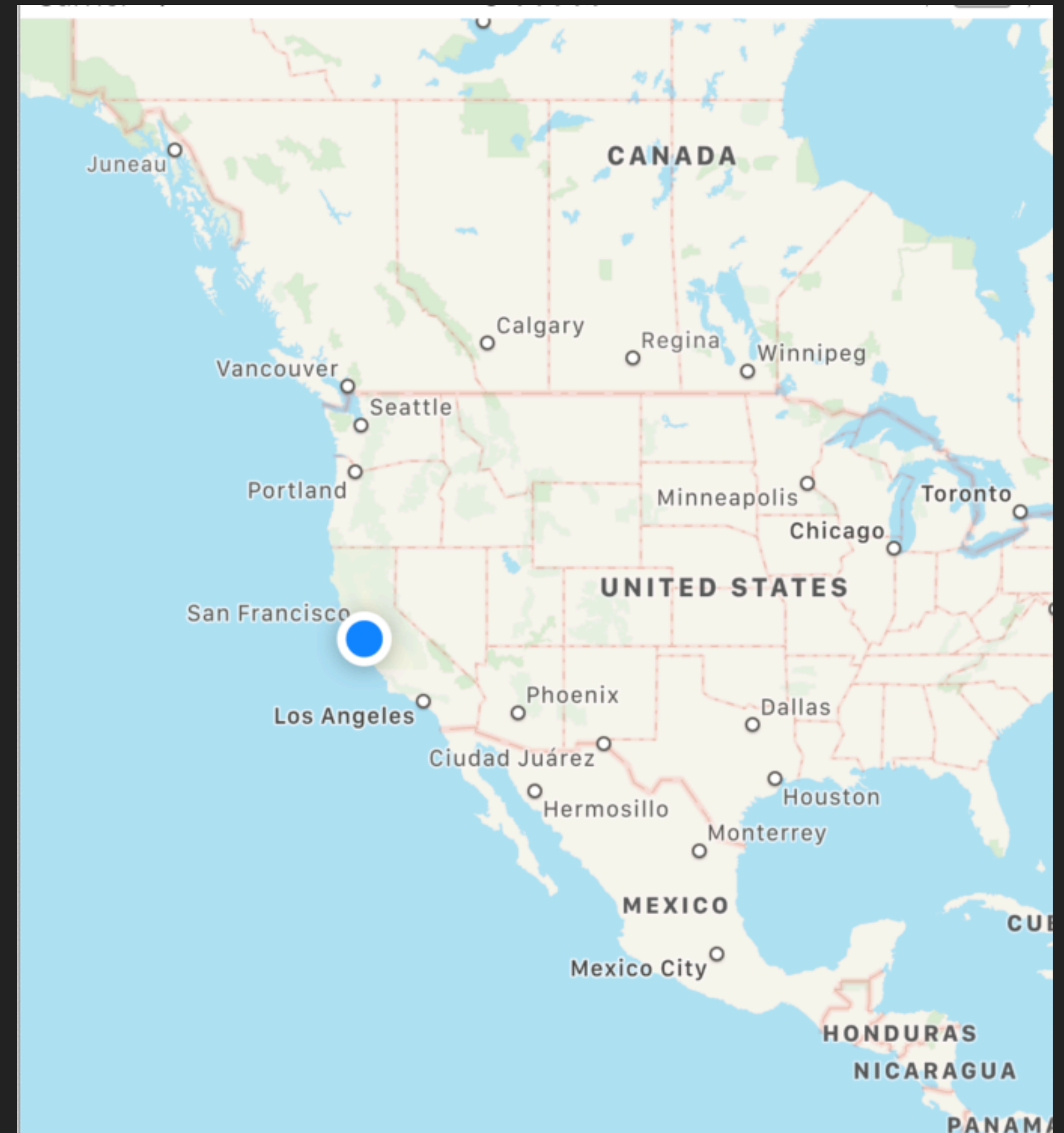
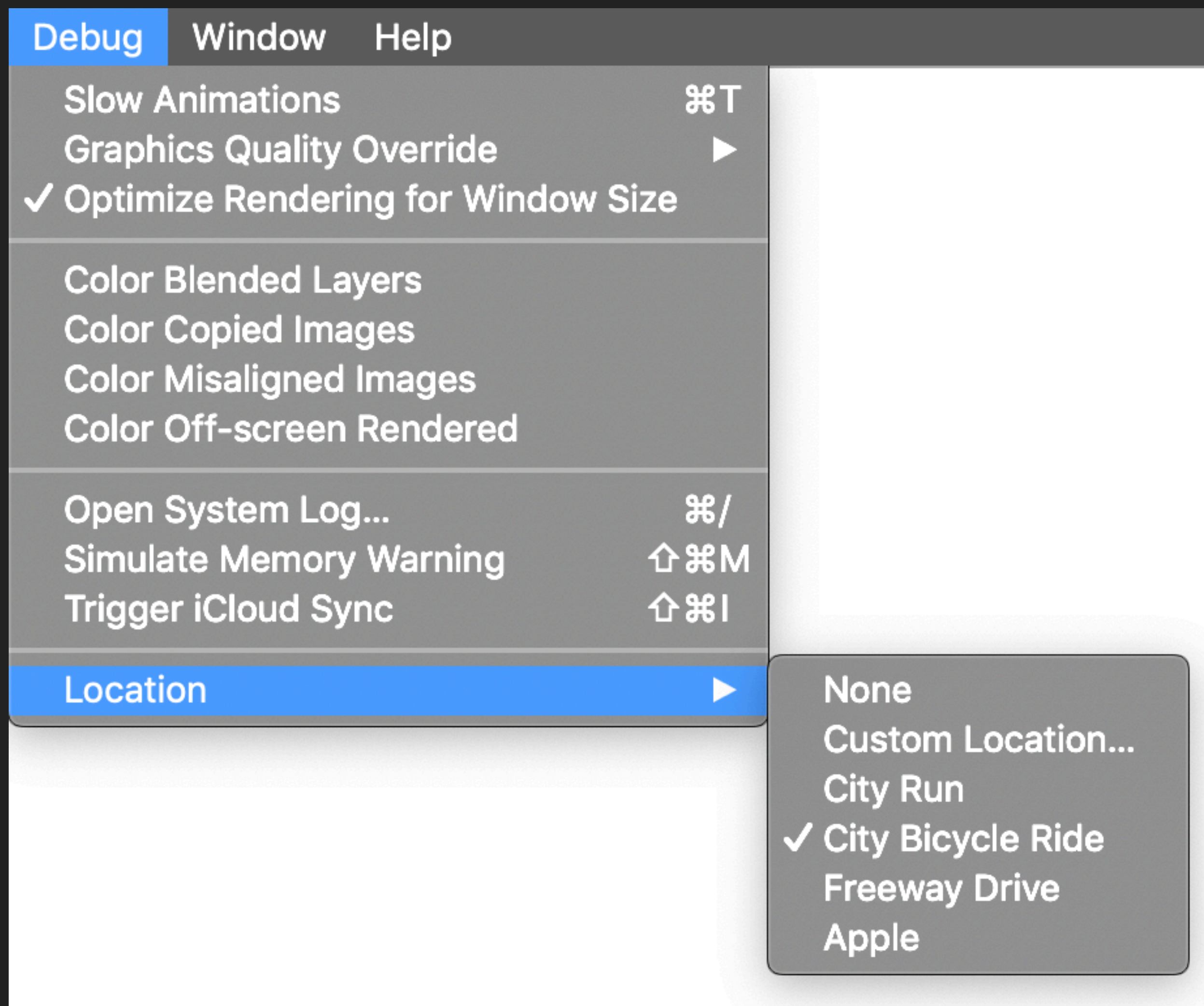
É possível selecionar o centro do mapa

Para centralizar na localização do usuário, é só colocar no método que recebemos a localização

Só não coloque sempre ou o usuário não conseguirá mover o mapa

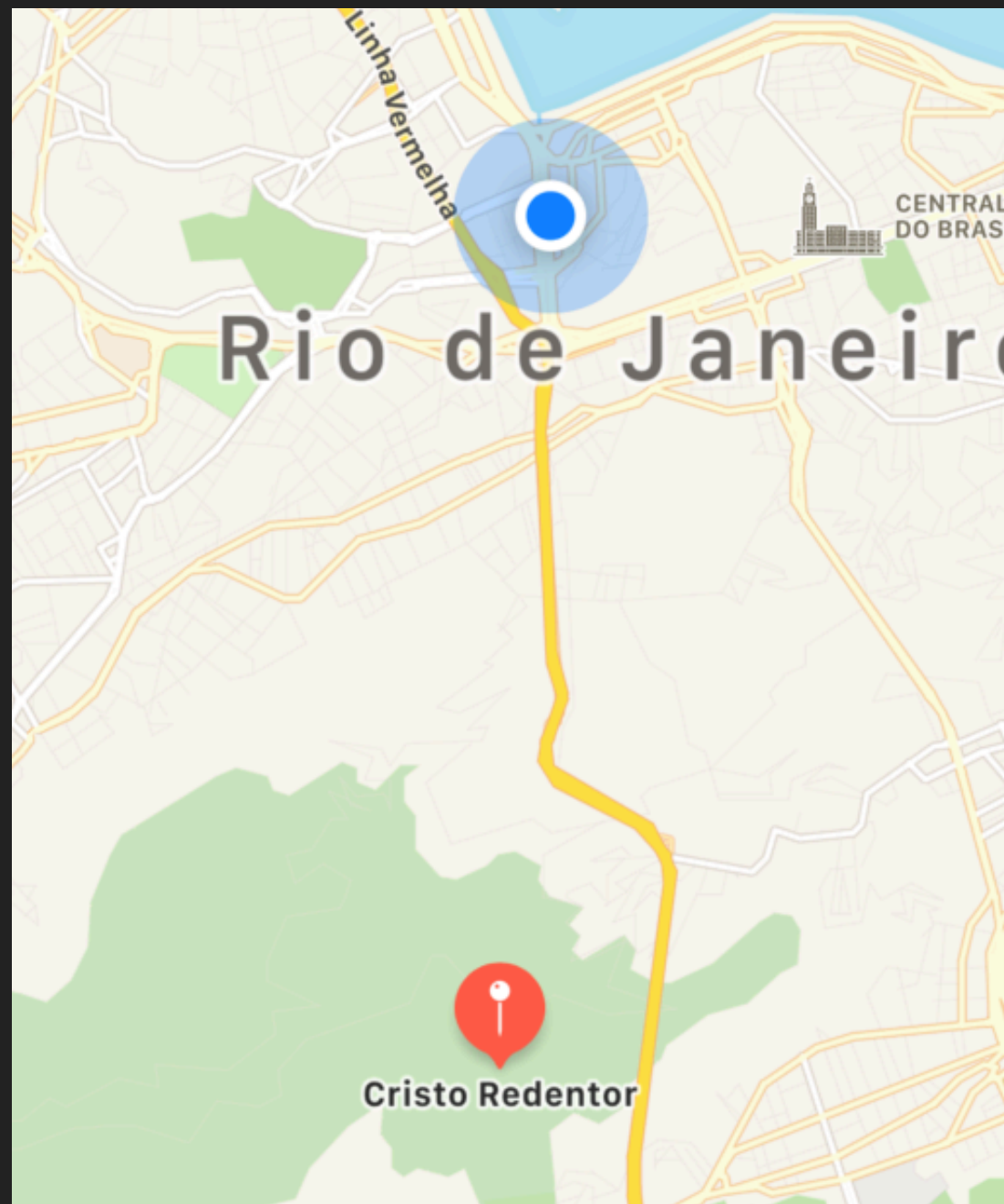
```
let copacabana = CLLocationCoordinate2D(latitude: -22.9523905, longitude: -43.2216452)
let span = MKCoordinateSpanMake(0.075, 0.075)
let region = MKCoordinateRegion(center: copacabana, span: span)
mapView.setRegion(region, animated: true)
```


Para testar a localização e movimento, selecione o simulador e o tipo de movimento:



Os pinos são MkPointAnnotation

```
let cristoRedentor = MKPointAnnotation()  
cristoRedentor.title = "Cristo Redentor"  
cristoRedentor.coordinate = CLLocationCoordinate2D(latitude: -22.951916, longitude: -43.2126759)  
mapView.addAnnotation(cristoRedentor)
```



COMO MUDAR O PIN?

MAPKIT

MKMapViewDelegate

```
class ViewController: UIViewController, CLLocationManagerDelegate, MKMapViewDelegate {
```


MAPKIT

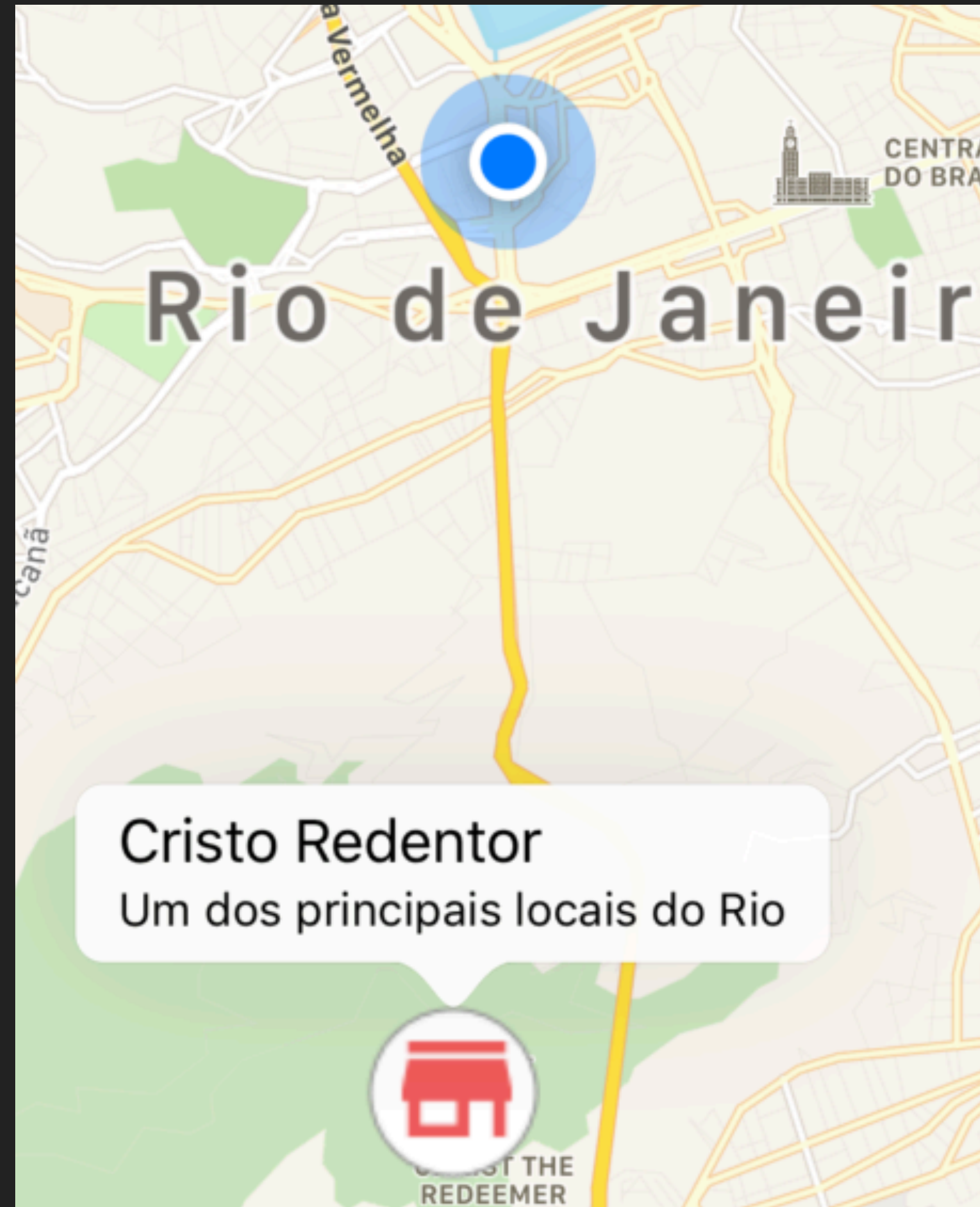
```
self.mapView.delegate = self
```

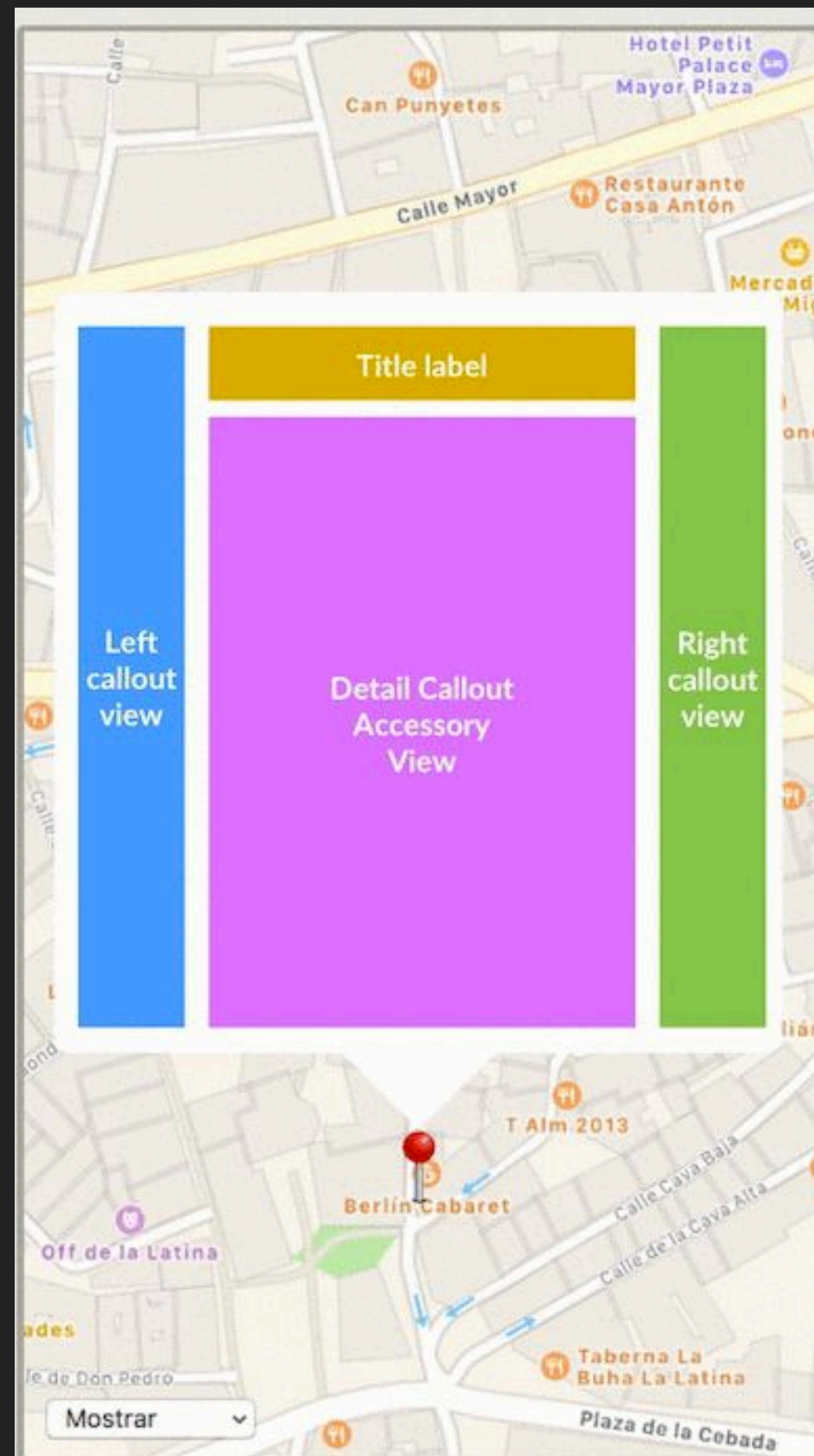
```
func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? {  
  
    let reuseIdentifier = "pin"  
    var annotationView = mapView.dequeueReusableAnnotationView(withIdentifier: reuseIdentifier)  
  
    if annotationView == nil {  
        annotationView = MKAnnotationView(annotation: annotation, reuseIdentifier: reuseIdentifier)  
        annotationView?.canShowCallout = true  
    } else {  
        annotationView?.annotation = annotation  
    }  
    annotationView?.image = UIImage(named: "loja-icon")  
  
    return annotationView  
}
```



MAPKIT

```
func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? {  
    guard !annotation.isKind(of: MKUserLocation.self) else {  
        return nil  
    }  
  
    let reuseIdentifier = "pin"  
    var annotationView = mapView.dequeueReusableAnnotationView(withIdentifier: reuseIdentifier)  
  
    if annotationView == nil {  
        annotationView = MKAnnotationView(annotation: annotation, reuseIdentifier: reuseIdentifier)  
        annotationView?.canShowCallout = true  
    } else {  
        annotationView?.annotation = annotation  
    }  
    annotationView?.image = UIImage(named: "loja-icon")  
  
    return annotationView  
}
```



CUSTOM CALLOUT



MIGRAÇÃO DO REALM

MIGRAÇÃO REALM

Agora que colocamos o mapa, precisamos mudar a nossa model para incluir os seguintes campos:

- ▶ Latitude
- ▶ Longitude
- ▶ Link para compartilhar

Posso então adicionar as seguinte properties na minha Model?

```
let latitude = RealmOptional<Double>()  
let longitude = RealmOptional<Double>()  
@objc dynamic var link: String?
```

NÃO SÓ

PODE, COMO

DEVE

**E VAI
FUNCIONAR?**

DEPENDE...

MIGRAÇÃO REALM

Temos três cenários:

- ▶ 1 Você nunca lançou uma versão do seu App, ele está em desenvolvimento:
 - ▶ 1.1 E você nunca executou ele
 - ▶ 1.2 Você já executou ele
- ▶ 2 Você já tem uma versão do seu App na loja e quer soltar uma atualização

MIGRAÇÃO REALM

1.1 Você **nunca** lançou uma versão do seu App, ele está em desenvolvimento e você **nunca** executou ele:

- ▶ Você pode só alterar a sua Model e executar que ela irá funcionar. Nunca houve um BD, então o Realm irá criar com essas alterações

MIGRAÇÃO REALM

1.2 Você **nunca** lançou uma versão do seu App, ele está em desenvolvimento e você **já** executou ele:

- ▶ Ao executar seu App no emulador (ou no Device) ele irá dar um erro
 - ▶ Existe uma versão do BD e você está tentando executar uma outra
- ▶ Como uma versão ainda não foi para loja, suponho que sua Model não esteja completa, então a coisa mais simples a se fazer é:
 - ▶ Desinstalar o App do emulador (ou do device)
 - ▶ Ao fazer uma alteração desse tipo até mesmo as pessoas que testam terão que desinstalar o App

MIGRAÇÃO REALM

2 Você **já** tem uma versão do seu App na loja e quer soltar uma atualização:

- ▶ Você terá de fazer uma migração no BD
 - ▶ Já existe um BD no celular dos seus usuário e você não quer que eles desinstalem e instalem novamente seu App;
 - ▶ O modelo do BD tem que ser atualizado para atender essas mudanças que fizemos.
- ▶ Vamos criar essas properties na nossa model e fazer a migração

MIGRAÇÃO REALM

- ▶ Para que nosso App não dê erro acessando o BD antigo com o modelo novo, precisamos acessar o BD e fazer a migração antes de qualquer coisa:
 - ▶ Vimos que o primeiro local que temos acesso ao código em nossa aplicação é no **AppDelegate** no método **didFinishLaunchingWithOptions**
 - ▶ Lá que faremos a migração
 - ▶ Precisamos alterar a configuração do Realm

```
Realm.Configuration.defaultConfiguration
```

MIGRAÇÃO REALM

Vamos criar a nova configuração

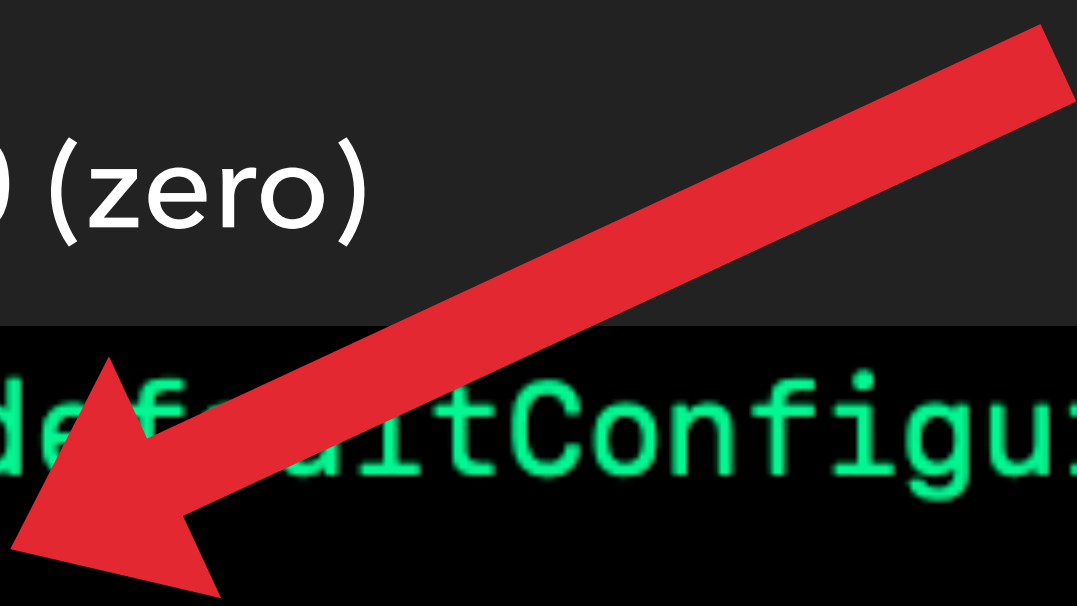


```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 1,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
    })  
})
```


MIGRAÇÃO REALM

Esse é a versão do BD que estará em vigor após a migração

A versão inicial é 0 (zero)




```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 1,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
    }  
})
```


MIGRAÇÃO REALM

Essa é a versão do BD que o App está nesse momento

No nosso caso é zero, mas se já tivéssemos feito uma migração ela seria um número diferente. Comparando o número da versão atual (oldSchemaVersion) com a versão que iremos (schemaVersion) que conseguimos fazer cada uma das migrações

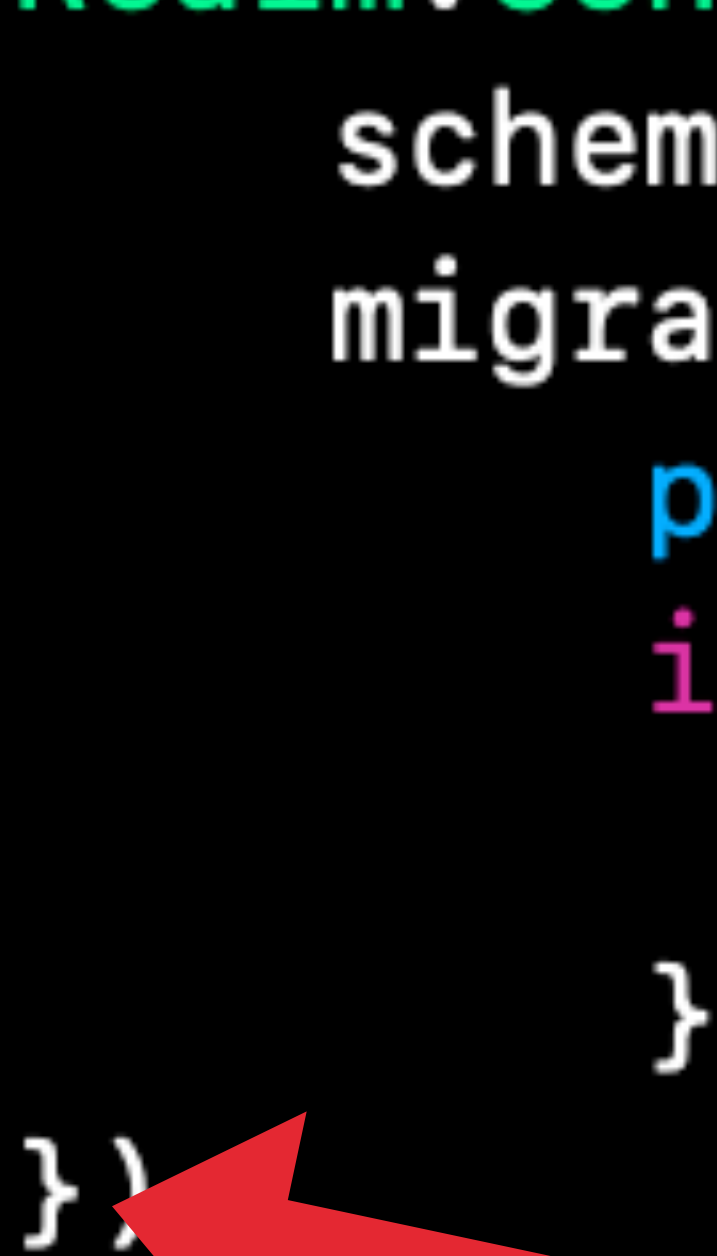


```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 1,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
    }  
})
```

MIGRAÇÃO REALM

Dentro dessa closure (migrationBlock) que iremos fazer as alterações necessárias


```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
  schemaVersion: 1,  
  migrationBlock: { migration, oldSchemaVersion in  
    print("oldSchemaVersion \(oldSchemaVersion)")  
    if (oldSchemaVersion < 1) {  
      print("Estou fazendo a migração")  
    }  
  })  
})
```



MIGRAÇÃO REALM

Sorte nossa que quando adicionamos algo (uma Model nova ou uma propriedade nova) sem alterar nada, não precisamos fazer nada

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 1,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
    }  
})
```



PRONTO!!!

E SE TIVÉSSEMOS
FEITO UMA
ALTERAÇÃO NA
NOSSA MODEL?

MIGRAÇÃO REALM

Que tal olharmos para essa alteração:

```
let latitude = RealmOptional<Double>()  
let longitude = RealmOptional<Double>()  
@objc dynamic var link: String?
```



```
@objc dynamic var latitude: String?  
@objc dynamic var longitude: String?  
@objc dynamic var link: String?
```

MIGRAÇÃO REALM

Digamos que nós publicamos duas versões do nosso app e estamos nos preparando para a terceira:

1ª versão - Classe loja que criamos inicialmente

- Versão do BD - 0

2ª versão - Classe loja com adição de latitude, longitude e link

- Versão do BD - 1

3ª versão - Classe loja que iremos alterar o tipo latitude e longitude de Double? para String?

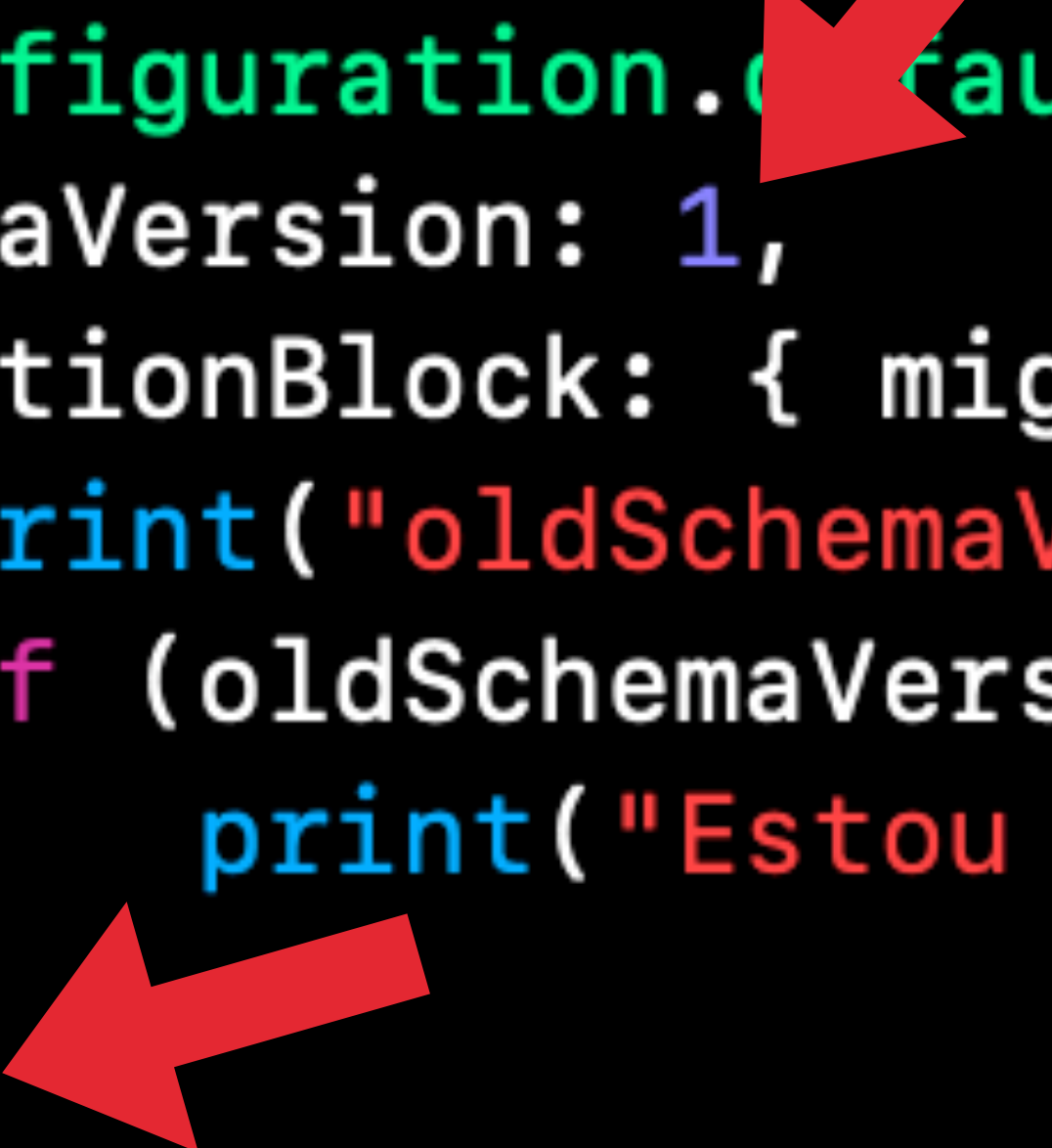
- Versão do BD - 2

MIGRAÇÃO REALM

A última versão que enviamos para a loja tinha a migração como abaixo.


Como alteramos nossa model e queremos enviar uma nova versão do Banco de Dados, teremos que fazer algumas alterações

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 1,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
    })
```



MIGRAÇÃO REALM

Vamos alterar a versão que o nosso BD irá estar após a migração:




```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 2,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
  
        if (oldSchemaVersion < 2) {  
        }  
    })
```


MIGRAÇÃO REALM

Iremos preparar o tratamento para o novo caso que criamos.

Dentro do if que iremos tratar a mudança do tipo dos objetos latitude e longitude

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 2,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
  
        if (oldSchemaVersion < 2) {  
              
        }  
    })
```

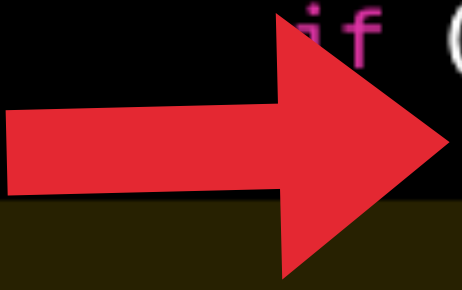

MIGRAÇÃO REALM

Com o `migration.enumeratedObjects` conseguimos o acesso aos objetos anteriores a migração e aos posteriores ao mesmo tempo

Essa closure será chamada várias vezes, passando por cada um dos objetos do BD

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(
    schemaVersion: 2,
    migrationBlock: { migration, oldSchemaVersion in
        print("oldSchemaVersion \(oldSchemaVersion)")
        if (oldSchemaVersion < 1) {
            print("Estou fazendo a migração")
        }

        if (oldSchemaVersion < 2) {
            migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in
                if let newObject = newObject, let oldObject = oldObject { 2 ⚠️ Immutable value
                }
            }
        }
    }
})
```




MIGRAÇÃO REALM

oldObject - É uma referência ao objeto atual (antes da migração) no qual latitude e longitude estão como **RealmOptional<Double>**

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(
    schemaVersion: 2,
    migrationBlock: { migration, oldSchemaVersion in
        print("oldSchemaVersion \(oldSchemaVersion)")
        if (oldSchemaVersion < 1) {
            print("Estou fazendo a migração")
        }

        if (oldSchemaVersion < 2) {
            migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in
                if let newObject = newObject, let oldObject = oldObject { 2 ⚠️ Immutable value
                }
            }
        }
    }
})
```




MIGRAÇÃO REALM

newObject - É uma referência ao objeto que queremos (após a migração) no qual latitude e longitude estão como **String**?

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(
    schemaVersion: 2,
    migrationBlock: { migration, oldSchemaVersion in
        print("oldSchemaVersion \(oldSchemaVersion)")
        if (oldSchemaVersion < 1) {
            print("Estou fazendo a migração")
        }

        if (oldSchemaVersion < 2) {
            migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in
                if let newObject = newObject, let oldObject = oldObject { 2 ⚠️ Immutable value
                }
            }
        }
    }
})
```



MIGRAÇÃO REALM


Faremos só uma verificação se eles não são nulos

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 2,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
  
        if (oldSchemaVersion < 2) {  
            migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
                if let newObject = newObject, let oldObject = oldObject { 2 ⚠️ Immutable value  
                }  
            }  
        }  
    }  
})
```

MIGRAÇÃO REALM

Tendo o objeto atual, vamos criar uma variável que recebe nossa latitude que é **RealmOptional<Double>**

```
if (oldSchemaVersion < 2) {  
    migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
        if let newObject = newObject, let oldObject = oldObject {  
            let oldLatitude = oldObject["latitude"] as? RealmOptional<Double>  
            newObject["latitude"] = "\(oldLatitude?.value ?? 0)"  
  
            let oldLongitude = oldObject["longitude"] as? RealmOptional<Double>  
            newObject["longitude"] = "\(oldLongitude?.value ?? 0)"  
        }  
    }  
}
```




MIGRAÇÃO REALM

Agora com a latitude **RealmOptional<Double>**, vamos transformar ela em uma String

Para acessar o valor do **RealmOptional** acessamos o value

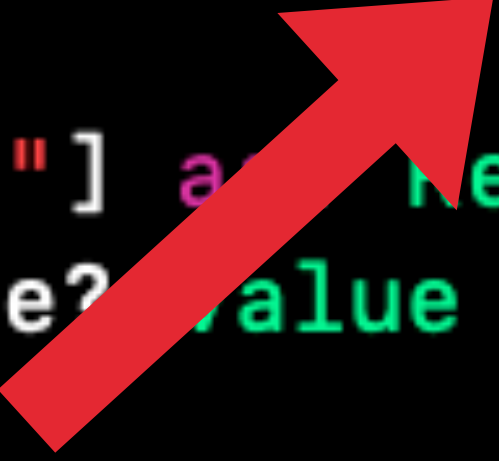
```
if (oldSchemaVersion < 2) {  
    migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
        if let newObject = newObject, let oldObject = oldObject {  
            let oldLatitude = oldObject["latitude"] as? RealmOptional<Double>  
            newObject["latitude"] = "\(oldLatitude?.value ?? 0)"  
  
            let oldLongitude = oldObject["longitude"] as? RealmOptional<Double>  
            newObject["longitude"] = "\(oldLongitude?.value ?? 0)"  
        }  
    }  
}
```



MIGRAÇÃO REALM

Caso o value seja nil, transforme em um zero

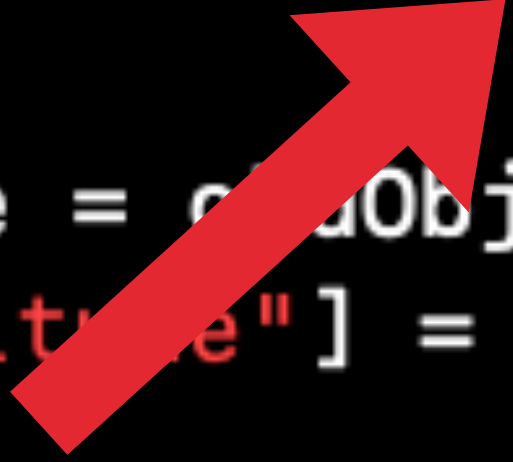
```
if (oldSchemaVersion < 2) {  
    migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
        if let newObject = newObject, let oldObject = oldObject {  
            let oldLatitude = oldObject["latitude"] as? RealmOptional<Double>  
            newObject["latitude"] = "\(oldLatitude?.value ?? 0)"  
  
            let oldLongitude = oldObject["longitude"] as? RealmOptional<Double>  
            newObject["longitude"] = "\(oldLongitude?.value ?? 0)"  
        }  
    }  
}
```



MIGRAÇÃO REALM

O resultado dessa expressão dentro de uma string


```
if (oldSchemaVersion < 2) {  
    migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
        if let newObject = newObject, let oldObject = oldObject {  
            let oldLatitude = oldObject["latitude"] as? RealmOptional<Double>  
            newObject["latitude"] = "\(oldLatitude?.value ?? 0)"  
  
            let oldLongitude = oldObject["longitude"] as? RealmOptional<Double>  
            newObject["longitude"] = "\(oldLongitude?.value ?? 0)"  
        }  
    }  
}
```



MIGRAÇÃO REALM

Populando a latitude do novo objeto que tem o formato String


```
if (oldSchemaVersion < 2) {  
    migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
        if let newObject = newObject, let oldObject = oldObject {  
            let oldLatitude = oldObject["latitude"] as? RealmOptional<Double>  
            newObject["latitude"] = "\(oldLatitude?.value ?? 0)"  
  
            let oldLongitude = oldObject["longitude"] as? RealmOptional<Double>  
            newObject["longitude"] = "\(oldLongitude?.value ?? 0)"  
        }  
    }  
}
```



MIGRAÇÃO REALM

O mesmo é feito para a longitude

```
if (oldSchemaVersion < 2) {  
    migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
        if let newObject = newObject, let oldObject = oldObject {  
            let oldLatitude = oldObject["latitude"] as? RealmOptional<Double>  
            newObject["latitude"] = "\(oldLatitude?.value ?? 0)"  
  
            let oldLongitude = oldObject["longitude"] as? RealmOptional<Double>  
            newObject["longitude"] = "\(oldLongitude?.value ?? 0)"  
        }  
    }  
}
```



PRONTO!!!

MIGRAÇÃO REALM

O Realm também possibilita a população de uma property por um conjunto de outras.

Digamos essa nova property

```
@objc dynamic var latlong: String?
```

Poderia ser populado pela concatenação da latitude antiga com a nova:

```
newObject["latlong"] = "\(String(describing: oldLatitude))  
    \(String(describing: oldLongitude))"
```

**EU PODERIA APAGAR OS
IF'S ANTERIORES UMA
VEZ QUE EU PUBLICO
UMA NOVA VERSÃO?**

MIGRAÇÃO REALM

Vamos analisar:

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 2,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
        if (oldSchemaVersion < 2) {  
            migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
                if let newObject = newObject, let oldObject = oldObject { 2 ⚠️ Immutable value  
                }  
            }  
        }  
    }  
})
```

CASO 1

MIGRAÇÃO REALM

Vamos analisar:

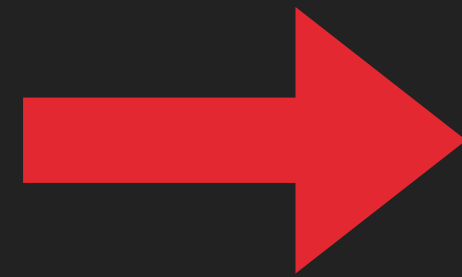
Instalou a
versão 0



MIGRAÇÃO REALM

Vamos analisar:

Instalou a
versão 0



oldSchema 0

Fez o update
do app v1

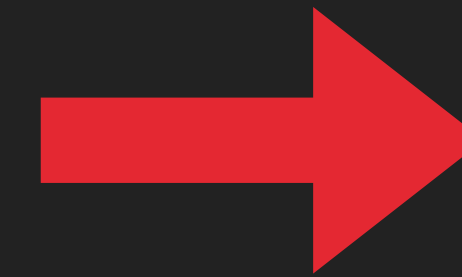
```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
    schemaVersion: 1,  
    migrationBlock: { migration, oldSchemaVersion in  
        print("oldSchemaVersion \(oldSchemaVersion)")  
        if (oldSchemaVersion < 1) {  
            print("Estou fazendo a migração")  
        }  
    })  
})
```

MIGRAÇÃO REALM

Vamos analisar:

Instalou a
versão 0

Fez o update
do app v1



oldSchema 1

Fez o update
do app v2

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
  schemaVersion: 2,  
  migrationBlock: { migration, oldSchemaVersion in  
    print("oldSchemaVersion \(oldSchemaVersion)")  
    if (oldSchemaVersion < 1) {  
      print("Estou fazendo a migração")  
    }  
  
    if (oldSchemaVersion < 2) {  
      migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
        if let newObject = newObject, let oldObject = oldObject { 2 ⚠️ Immutable value  
        }  
      }  
    }  
  }  
})
```

CASO 2

MIGRAÇÃO REALM

Vamos analisar:

Instalou a
versão 0



MIGRAÇÃO REALM

Vamos analisar:

Instalou a
versão 0

Pulou a versão 1

Fez o update
do app v2

oldSchema 0

```
Realm.Configuration.defaultConfiguration = Realm.Configuration(  
  schemaVersion: 2,  
  migrationBlock: { migration, oldSchemaVersion in  
    print("oldSchemaVersion \(oldSchemaVersion)")  
    if (oldSchemaVersion < 1) {  
      print("Estou fazendo a migração")  
    }  
  
    if (oldSchemaVersion < 2) {  
      migration.enumerateObjects(ofType: Loja.className()) { oldObject, newObject in  
        if let newObject = newObject, let oldObject = oldObject { 2 ⚠️ Immutable value  
        }  
      }  
    }  
  }  
})
```