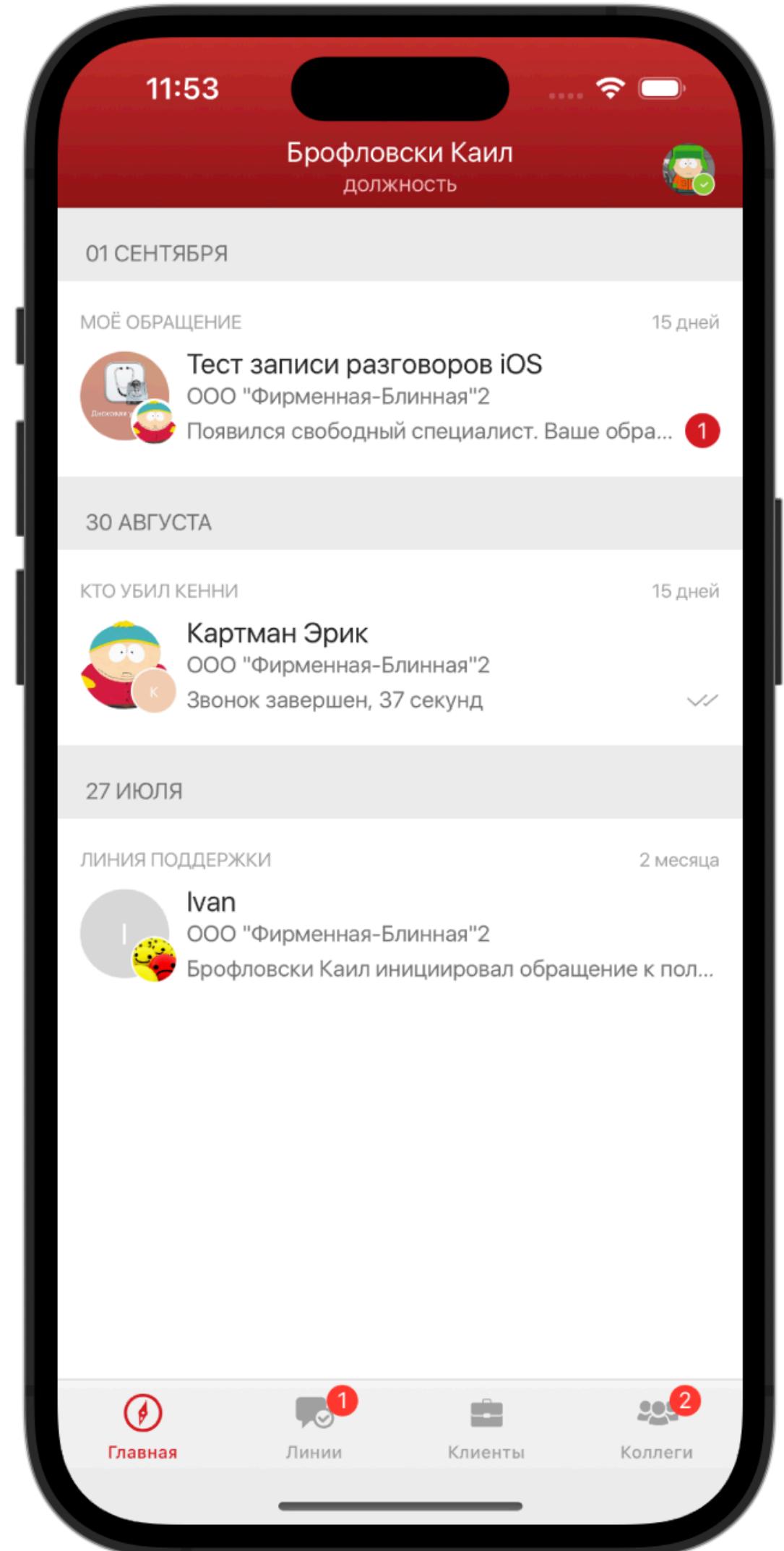


# Опыт разработки iOS Применение VIPER



Сергей Долгих  
iOS разработчик,  
1С-Коннект

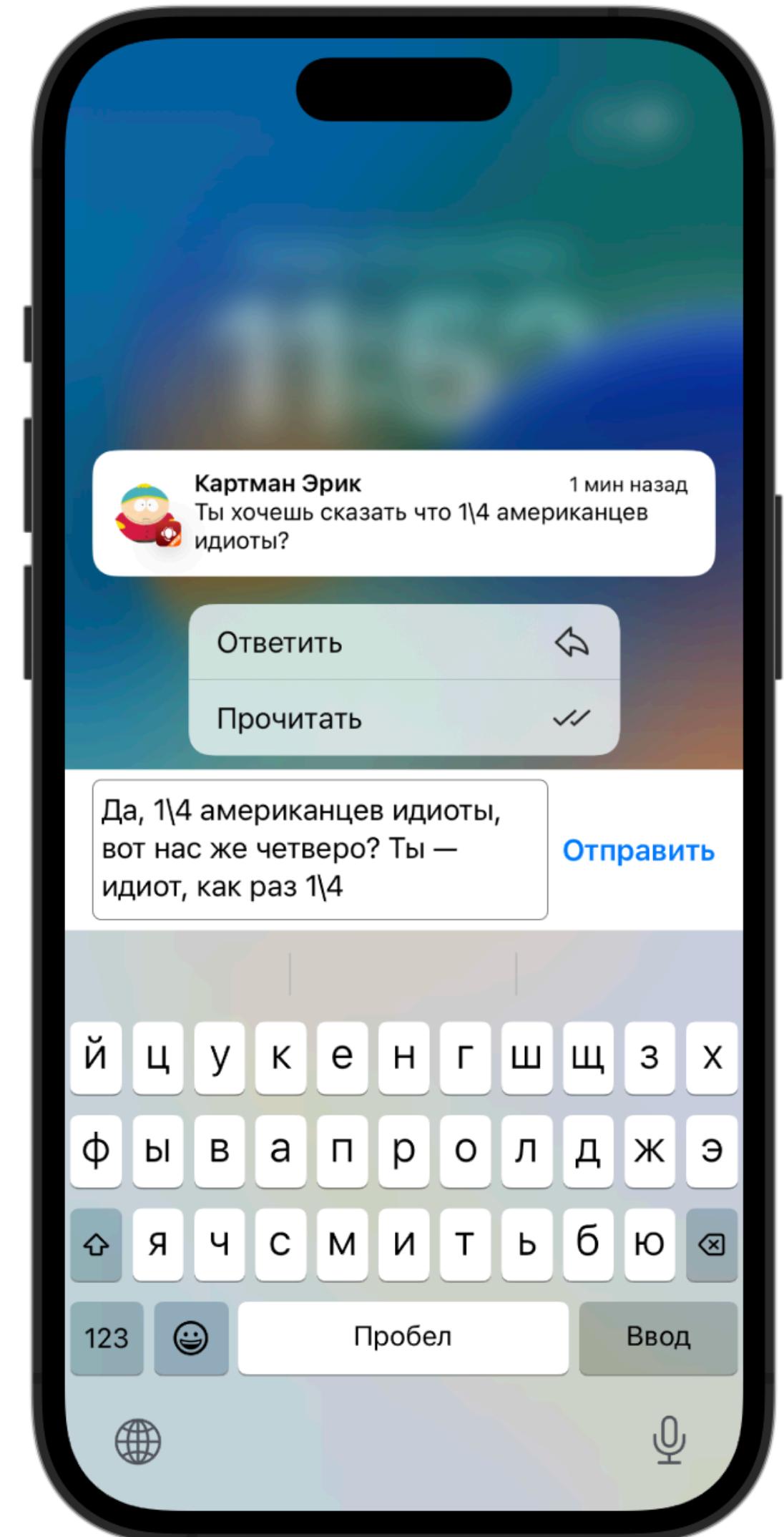
# MainApp



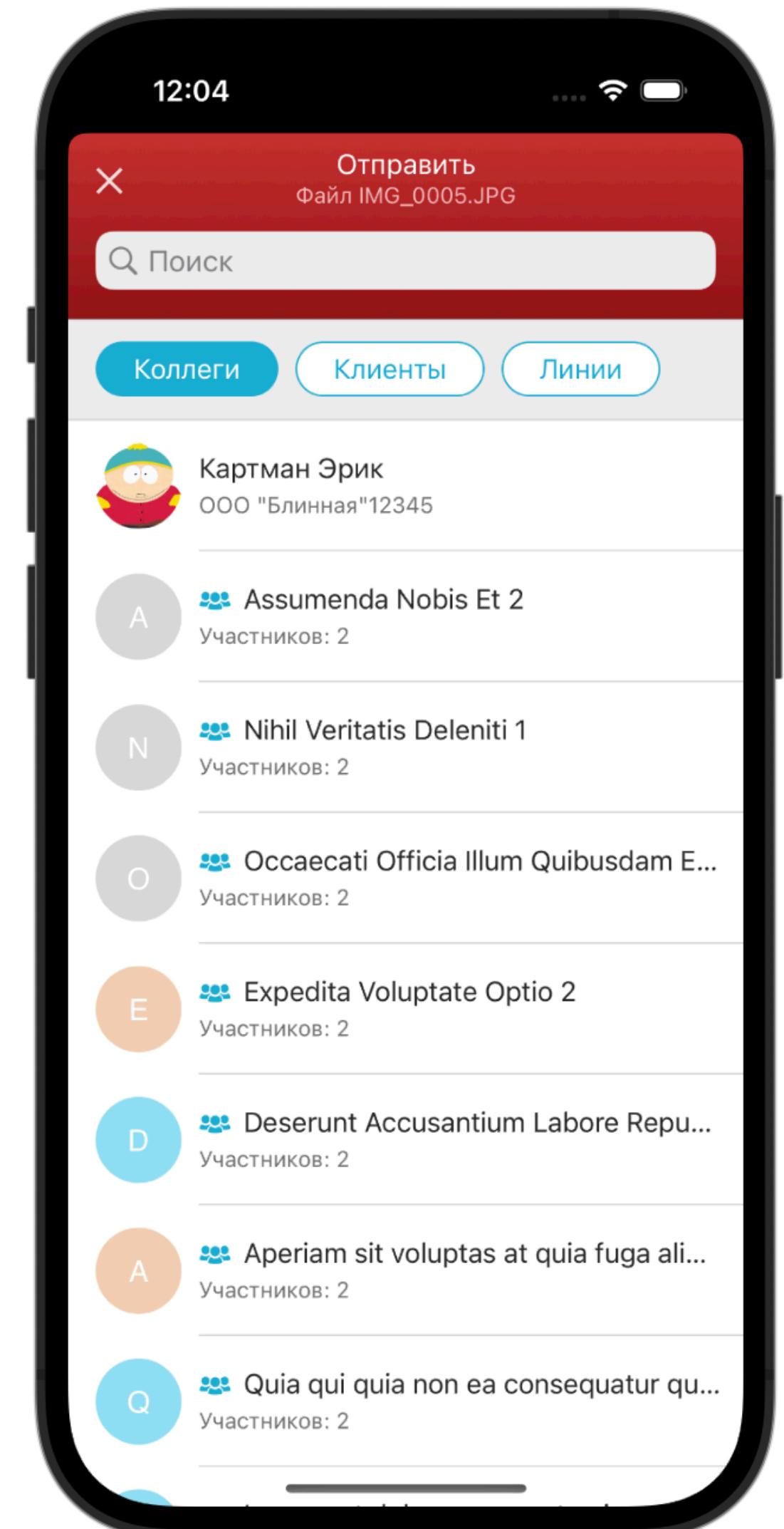
# NotificationServiceExtension



# NotificationContentExtension



# ShareExtension



# IntentsExtension



# VIPER

Что это такое?



# Архитектура

- разделение обязанностей
- упрощение тестирования

# VIPER

Что это такое?

**V**iew

**I**nteractor

**P**resenter

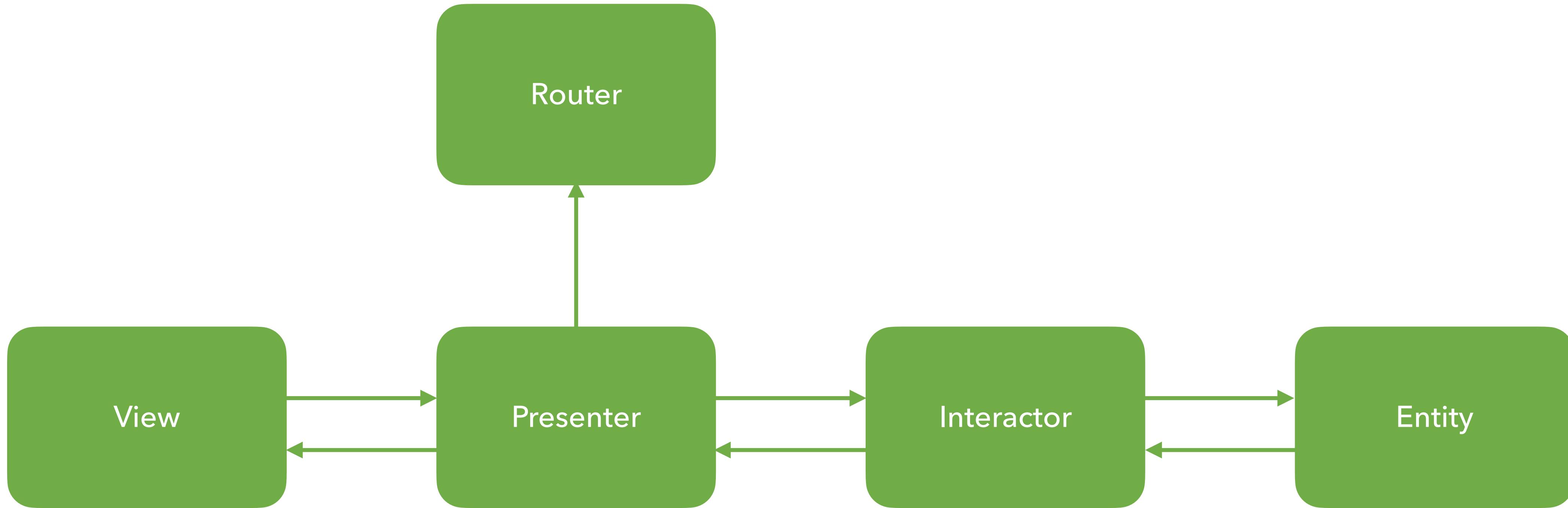
**E**ntity

**R**outer



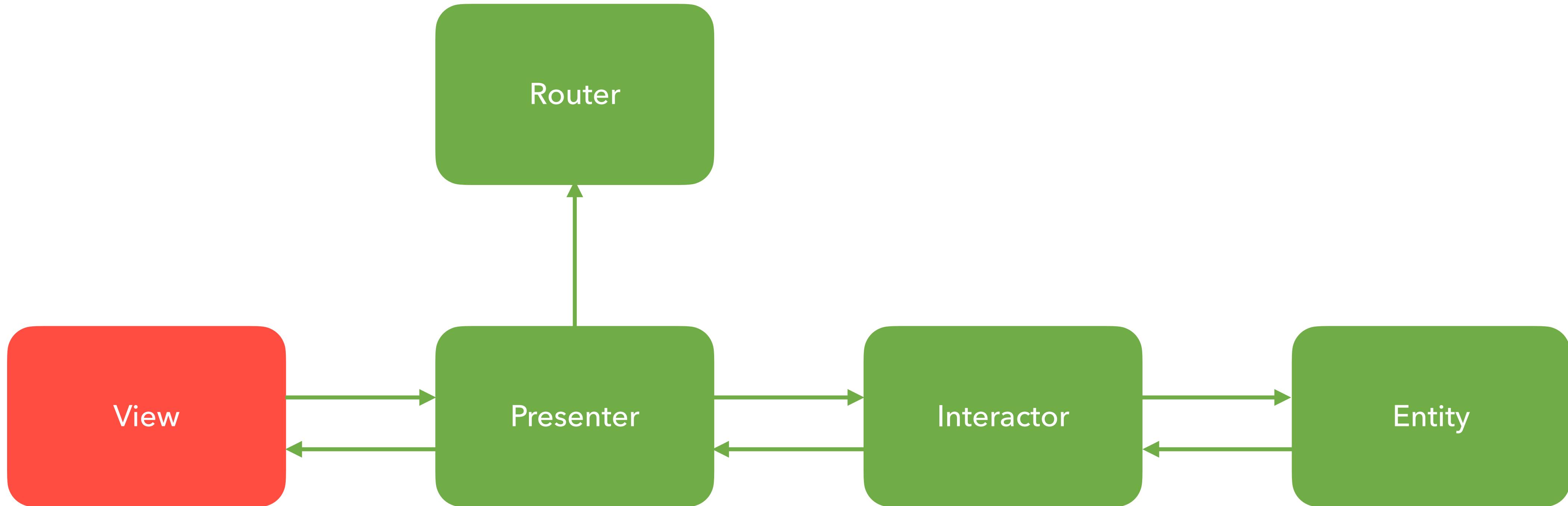
# VIPER

Что это такое?



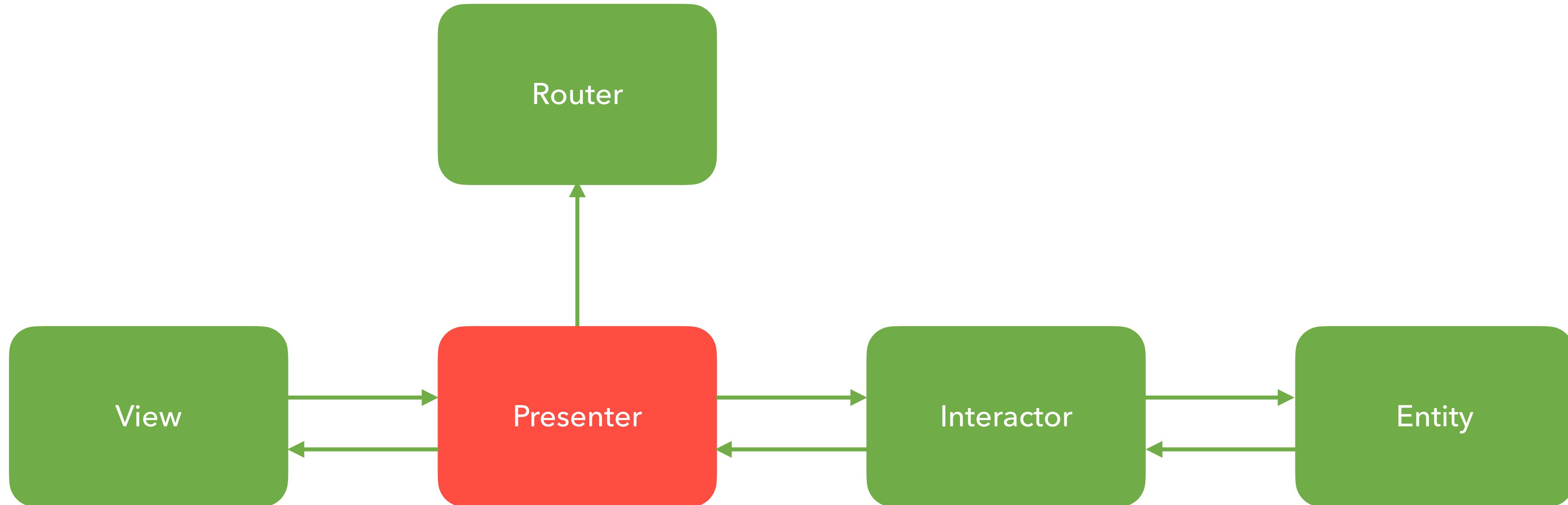
# VIPER

Что это такое?



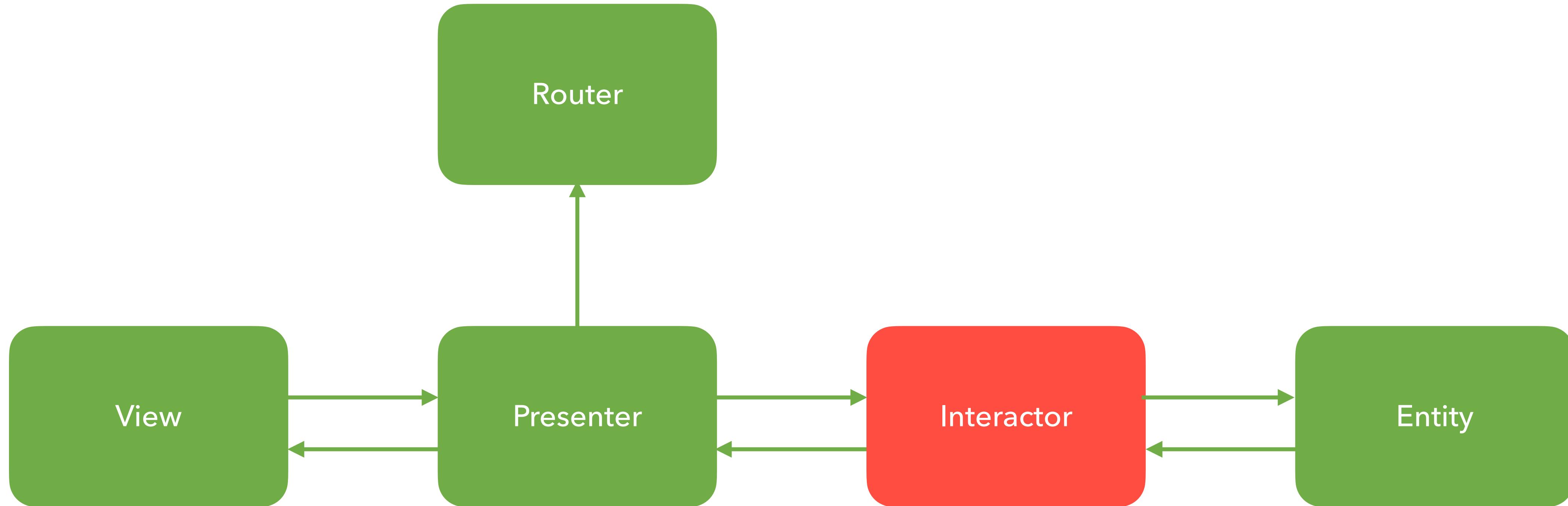
# VIPER

Что это такое?



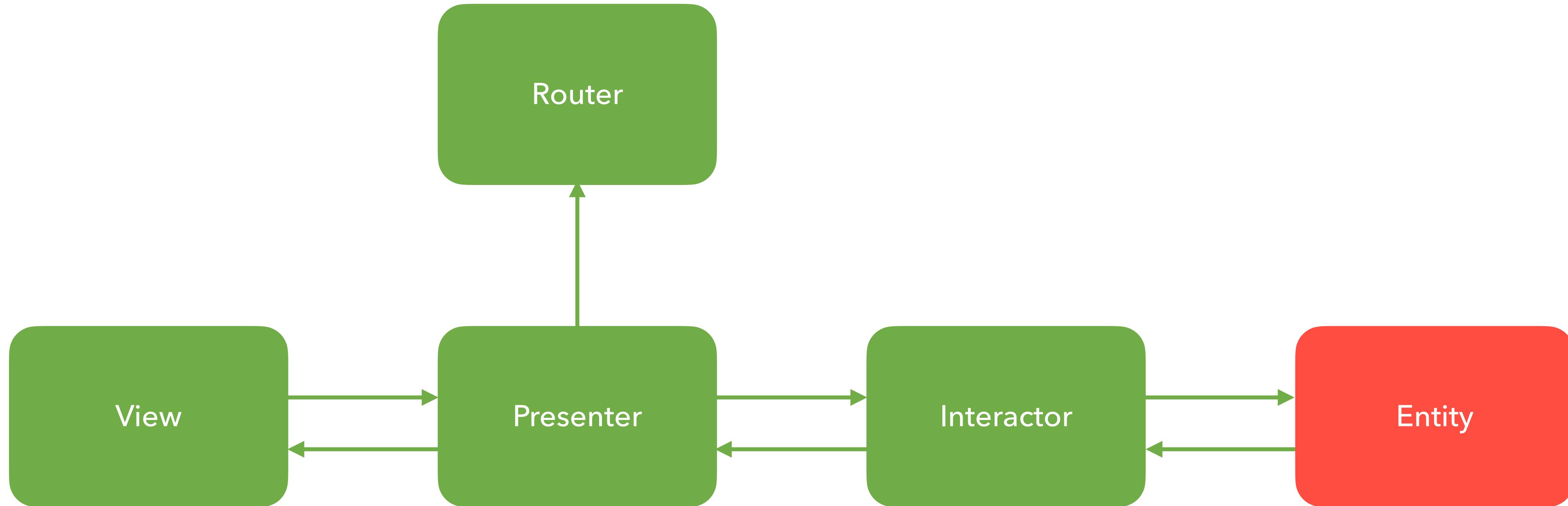
# VIPER

Что это такое?



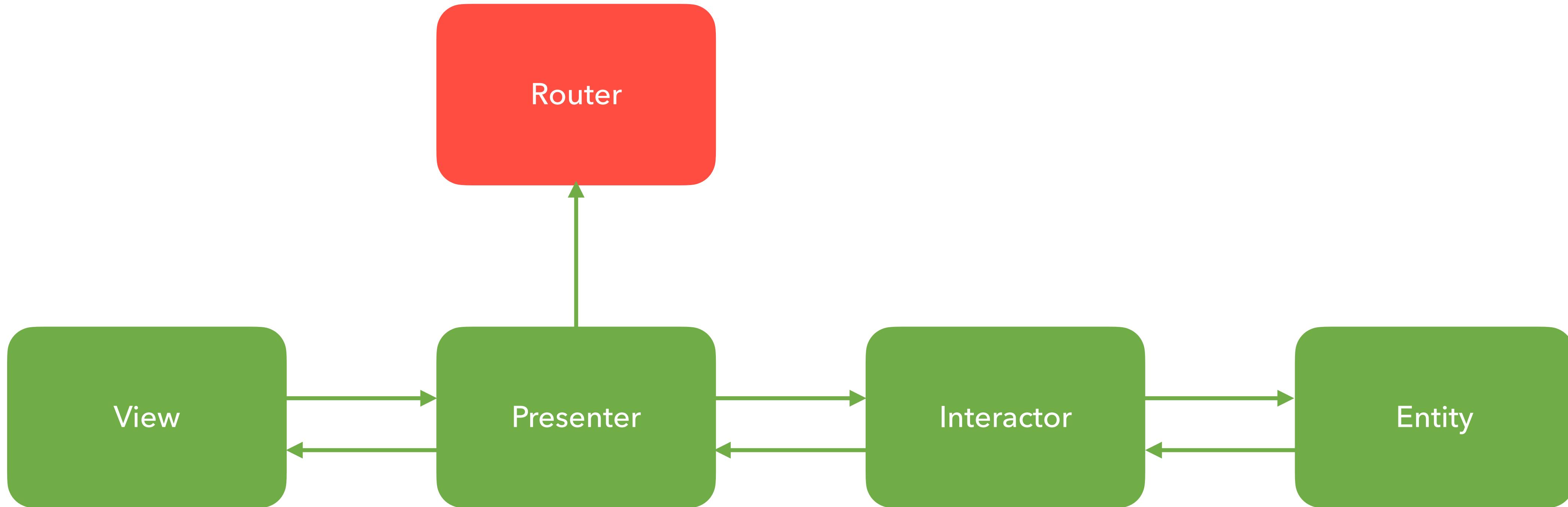
# VIPER

Что это такое?



# VIPER

Что это такое?



# Entity

DataStorage

DiskCacheManager

ApiService

UpdateService

XmppService

MessageService

ParseMessageService

ParseEmojiService

FileService

AuthorizeService

AnalyticsManager

Logger

CallService

PushKitService

UserNotificationService

KeyboardService

LinkService

Keychain

CommunicationInteractor

AudioService

# Entity

DataStorage

DiskCacheManager

ApiService

UpdateService

XmppService

MessageService

ParseMessageService

ParseEmojiService

FileService

AuthorizeService

AnalyticsManager

Logger

CallService

PushKitService

UserNotificationService

KeyboardService

LinkService

Keychain

CommunicationInteractor

AudioService

# Entity

DataStorage

DiskCacheManager

ApiService

UpdateService

XmppService

MessageService

ParseMessageService

ParseEmojiService

FileService

AuthorizeService

AnalyticsManager

Logger

CallService

PushKitService

UserNotificationService

KeyboardService

LinkService

Keychain

CommunicationInteractor

AudioService

# Entity

DataStorage

DiskCacheManager

ApiService

UpdateService

XmppService

MessageService

ParseMessageService

ParseEmojiService

FileService

AuthorizeService

AnalyticsManager

Logger

CallService

PushKitService

UserNotificationService

KeyboardService

LinkService

Keychain

CommunicationInteractor

AudioService

# Entity

DataStorage

DiskCacheManager

ApiService

UpdateService

XmppService

MessageService

ParseMessageService

ParseEmojiService

FileService

AuthorizeService

AnalyticsManager

Logger

CallService

PushKitService

UserNotificationService

KeyboardService

LinkService

Keychain

CommunicationInteractor

AudioService

# Router



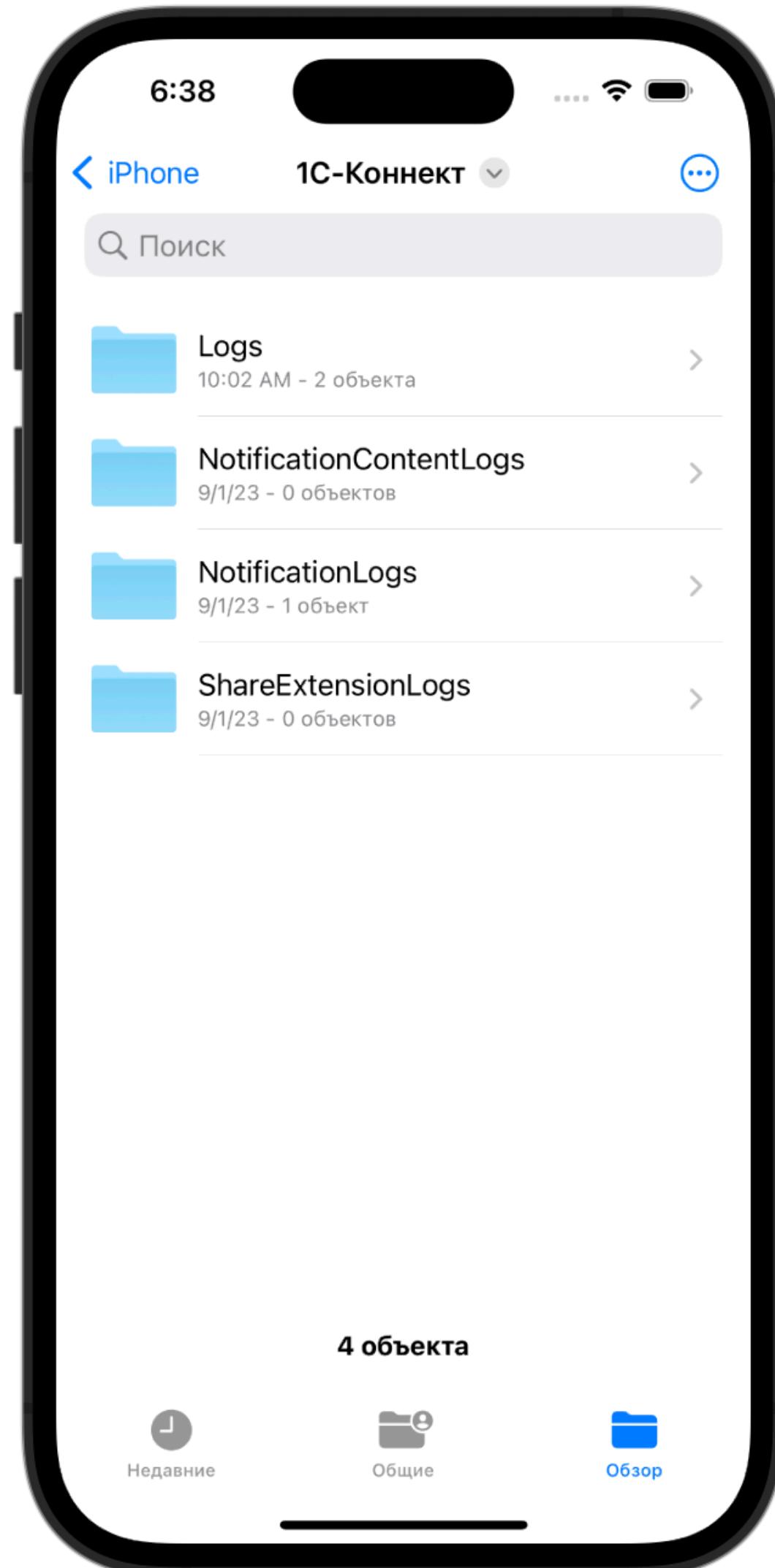
```
1 func presentLineInfo(lineId: String) {  
2     let infoVC = LineInfoModuleAssembly.assembleModule(lineId: lineId)  
3     let navigationController = BaseNavigationController(rootViewController: infoVC)  
4  
5     viewController.navigationController?.present(navigationController, animated: true)  
6 }  
7  
8 func presentCall(call: Call) {  
9     RootRouter.openCallView(call: call)  
10 }
```

# Router



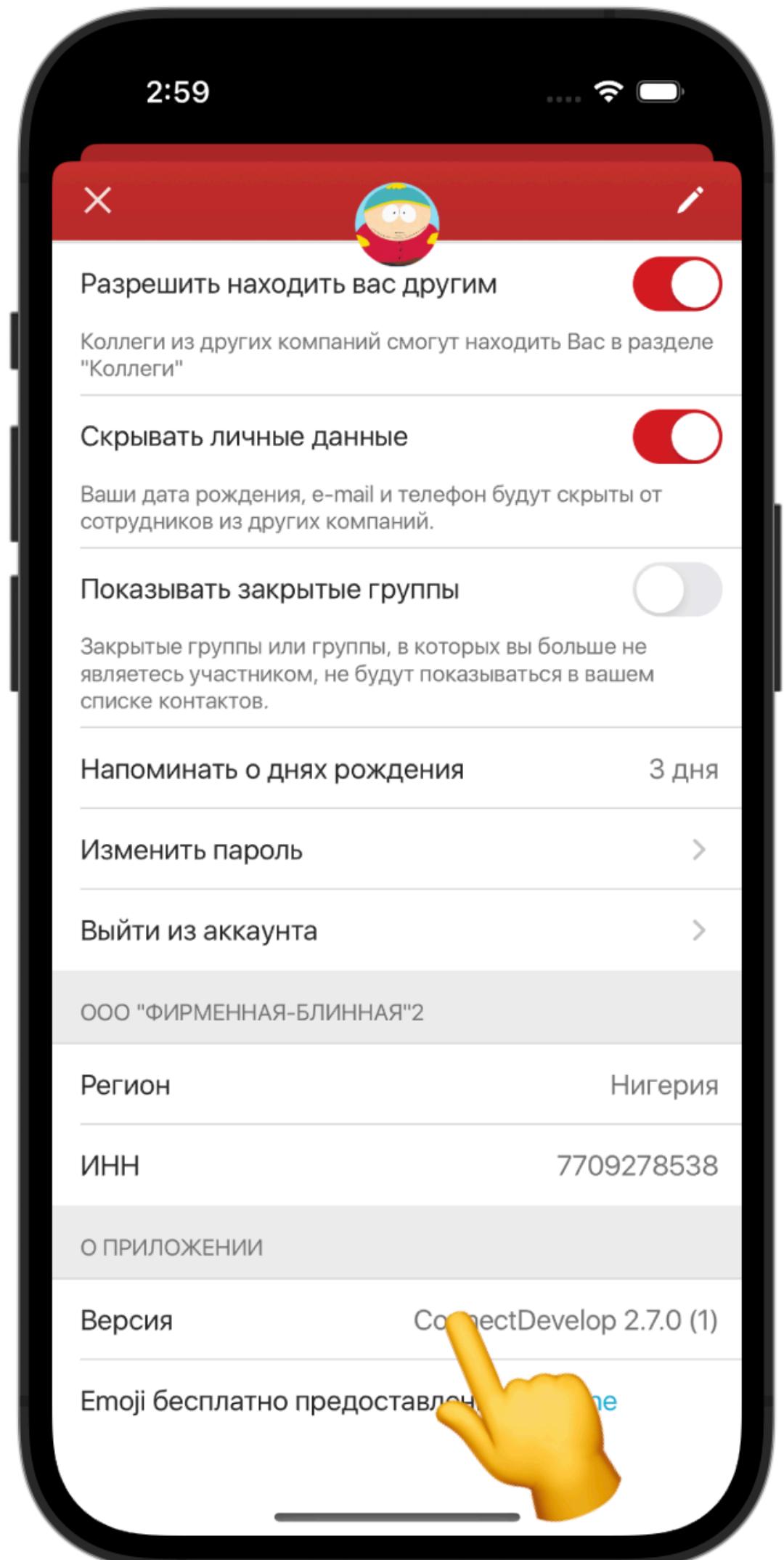
```
1 func presentLineInfo(lineId: String) {  
2     let infoVC = LineInfoModuleAssembly.assembleModule(lineId: lineId)  
3     let navigationController = BaseNavigationController(rootViewController: infoVC)  
4  
5     viewController.navigationController?.present(navigationController, animated: true)  
6 }  
7  
8 func presentCall(call: Call) {  
9     RootRouter.openCallView(call: call)  
10 }
```

# Пример



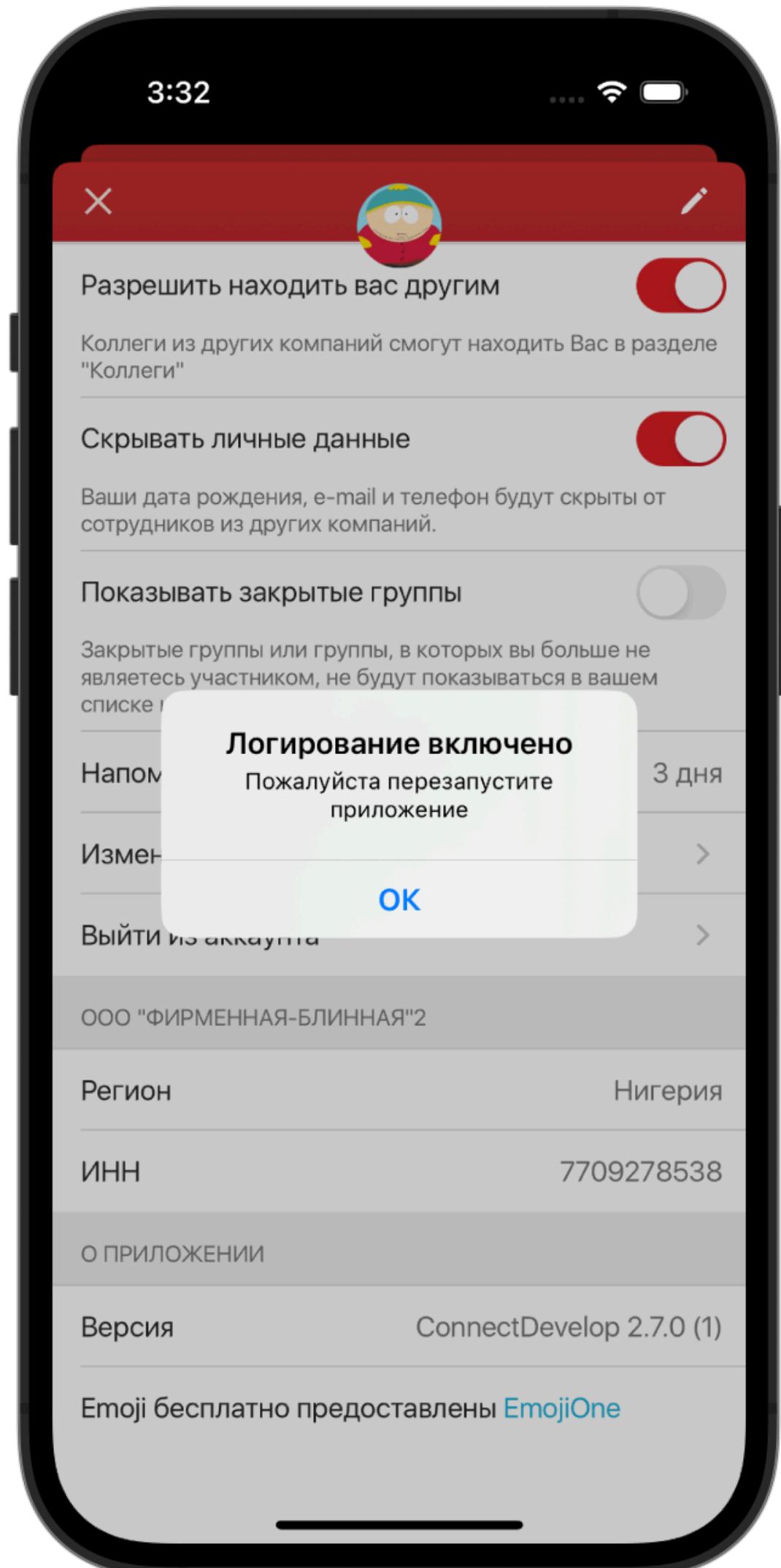
Включение сохранение логов  
приложения в файл

# Пример



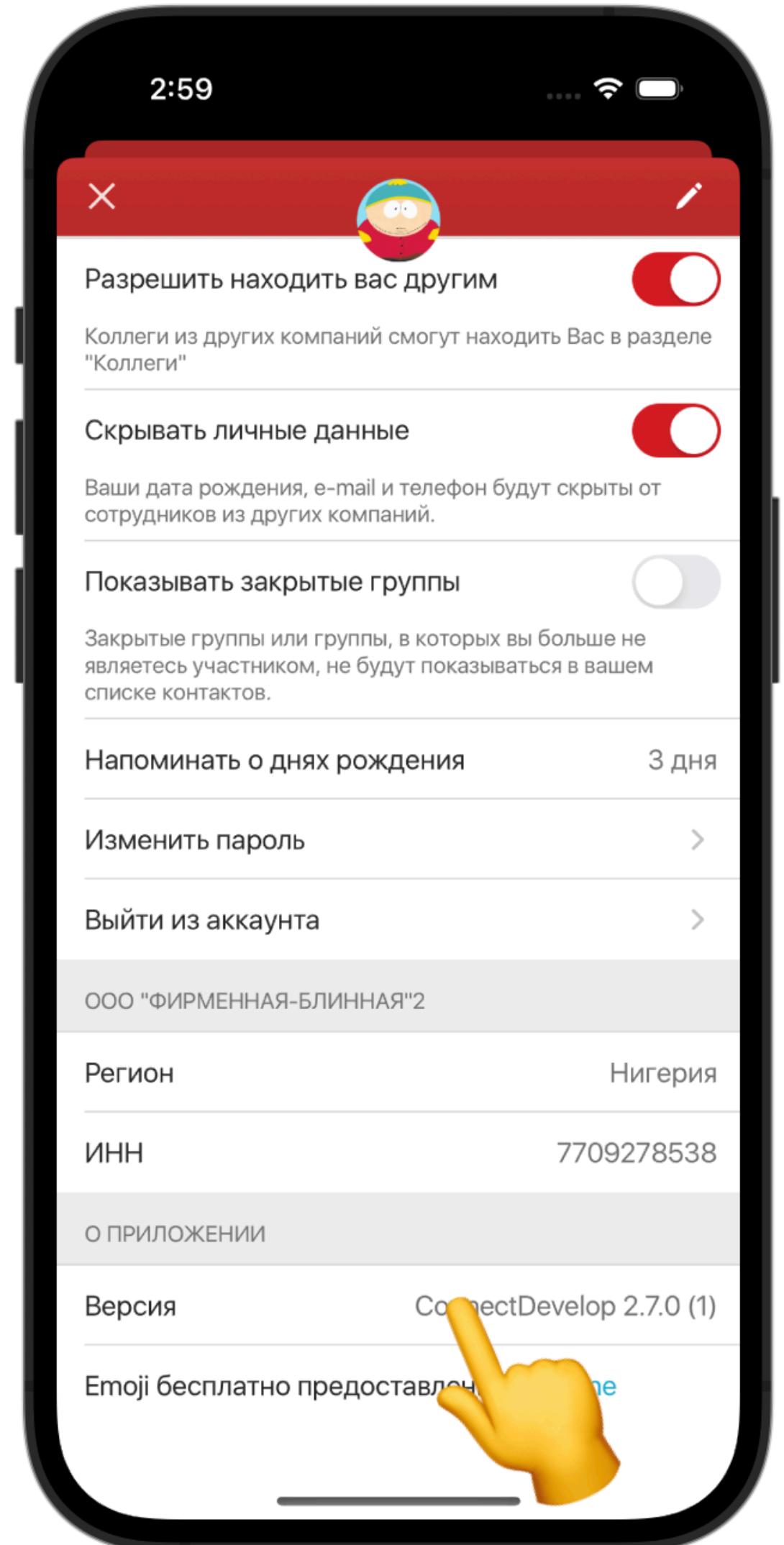
- Пользователь долго удерживает палец на версии приложения

# Пример



- Пользователь долго удерживает палец на версии приложения
- Появляется уведомление о включении или выключении логирования

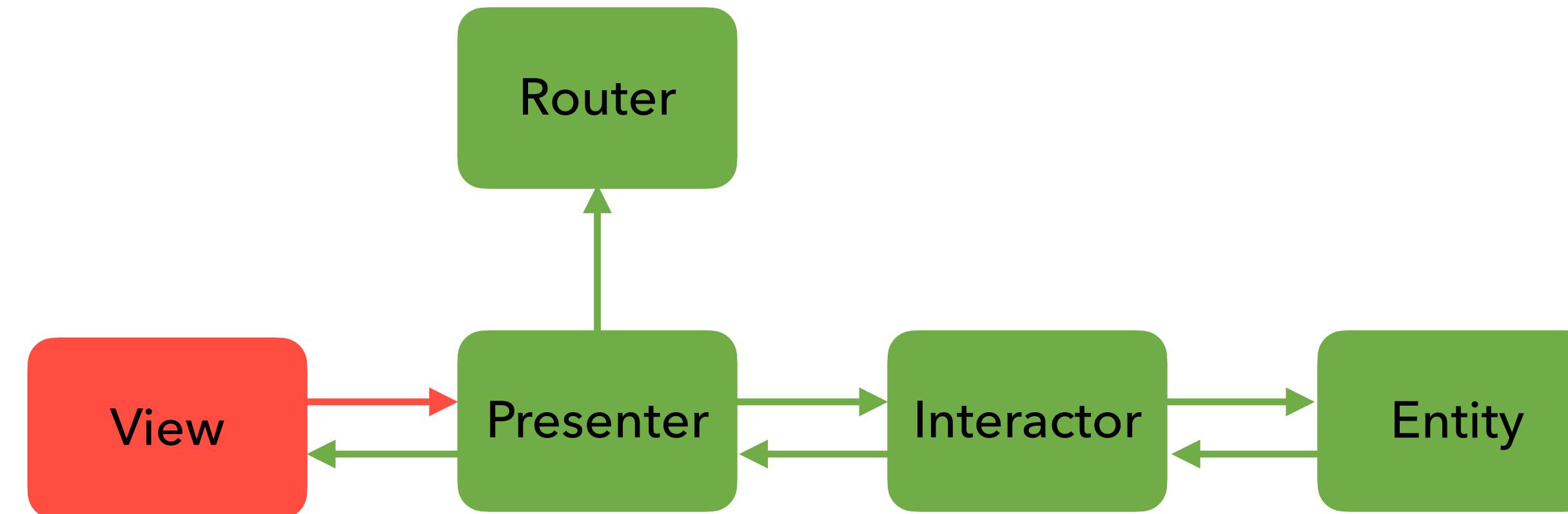
# Под капотом



# Под капотом



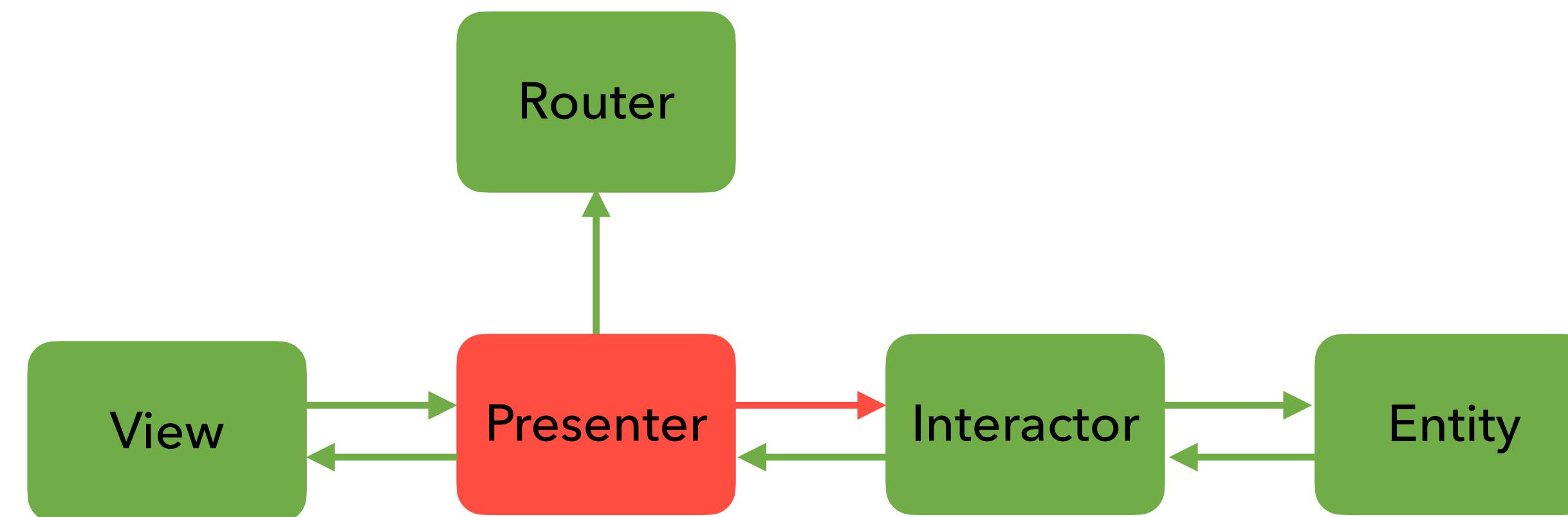
```
1 func longPressGestureRecognized(_ sender: UILongPressGestureRecognizer) {  
2     guard sender.state == .began else { return }  
3  
4     let indexPath = tableView.indexPathForRow(at: sender.location(in: tableView))  
5     guard  
6         let indexPath,  
7         indexPath.section <= sections.count,  
8         indexPath.row <= sections[indexPath.section].rows.count  
9     else { return }  
10  
11    let cellType = sections[indexPath.section].rows[indexPath.row].type  
12    presenterDispatch.async { $0.didLongPressedCell(cellType) }  
13 }
```



# Под капотом

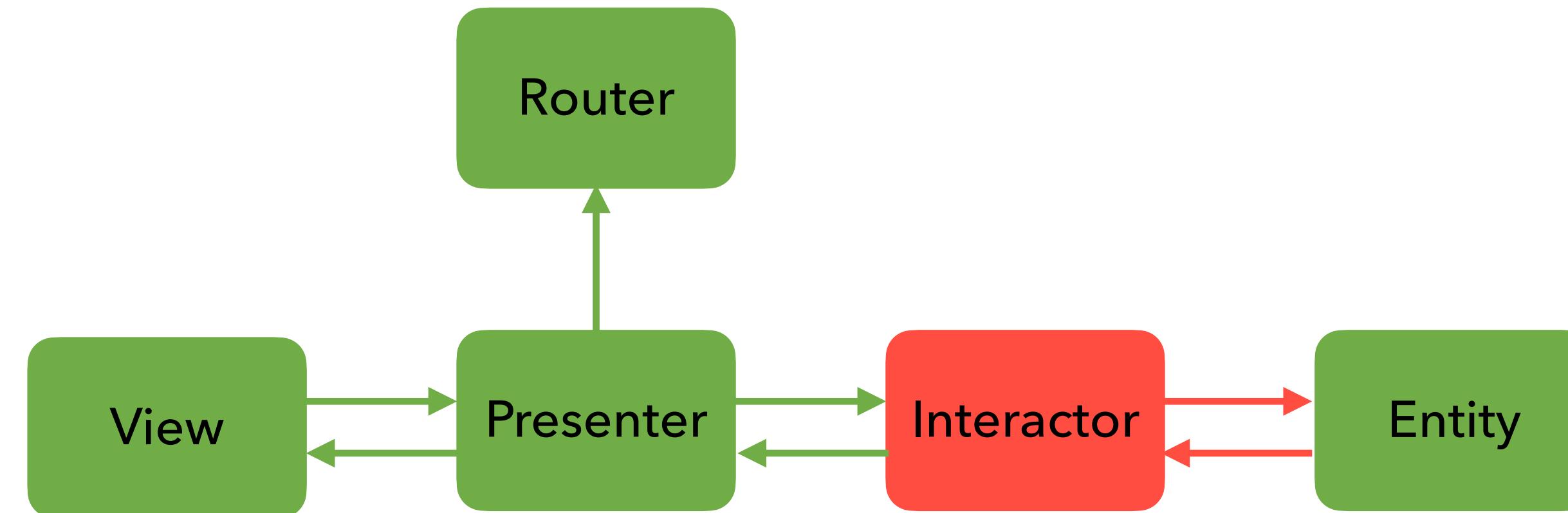


```
1 func didLongPressedCell(_ cellType: ProfileCellType) {  
2     guard cellType == .appVersion else { return }  
3     interactorDispatch.async { $0.changeForceLogging() }  
4 }
```



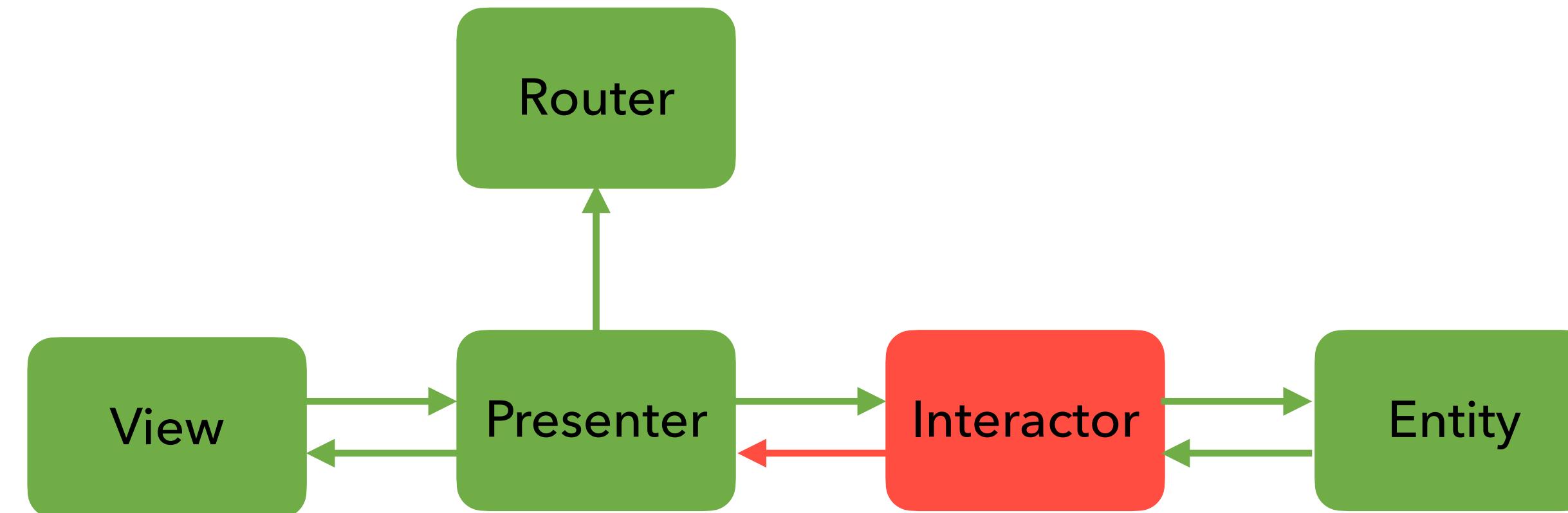
# Под капотом

```
1 func changeForceLogging() {  
2     let key = KeychainKeys.isEnabledLogging.keyString()  
3     let isEnabledLog = debugKeychain.bool(forKey: key) ?? false  
4  
5     let isSaved = debugKeychain.set(!isEnabledLog, forKey: key, withAccessibility: .always)  
6     if isSaved {  
7         presenterDispatch.async { $0.forceLogging(enable: !isEnabledLog) }  
8     } else {  
9         presenterDispatch.async { $0.forceLogging(enable: nil) }  
10        Logger.error("Failed change force logging")  
11    }  
12 }
```



# Под капотом

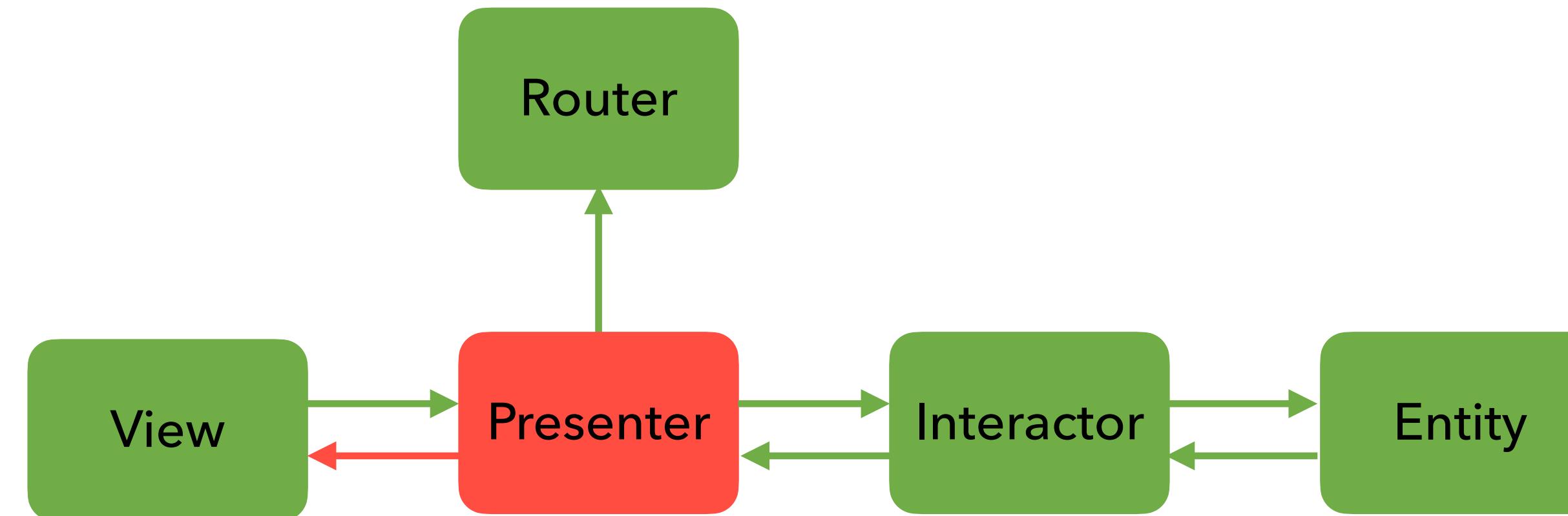
```
1 func changeForceLogging() {  
2     let key = KeychainKeys.isEnabledLogging.keyString()  
3     let isEnabledLog = debugKeychain.bool(forKey: key) ?? false  
4  
5     let isSaved = debugKeychain.set(!isEnabledLog, forKey: key, withAccessibility: .always)  
6     if isSaved {  
7         presenterDispatch.async { $0.forceLogging(enable: !isEnabledLog) }  
8     } else {  
9         presenterDispatch.async { $0.forceLogging(enable: nil) }  
10        Logger.error("Failed change force logging")  
11    }  
12 }
```



# Под капотом

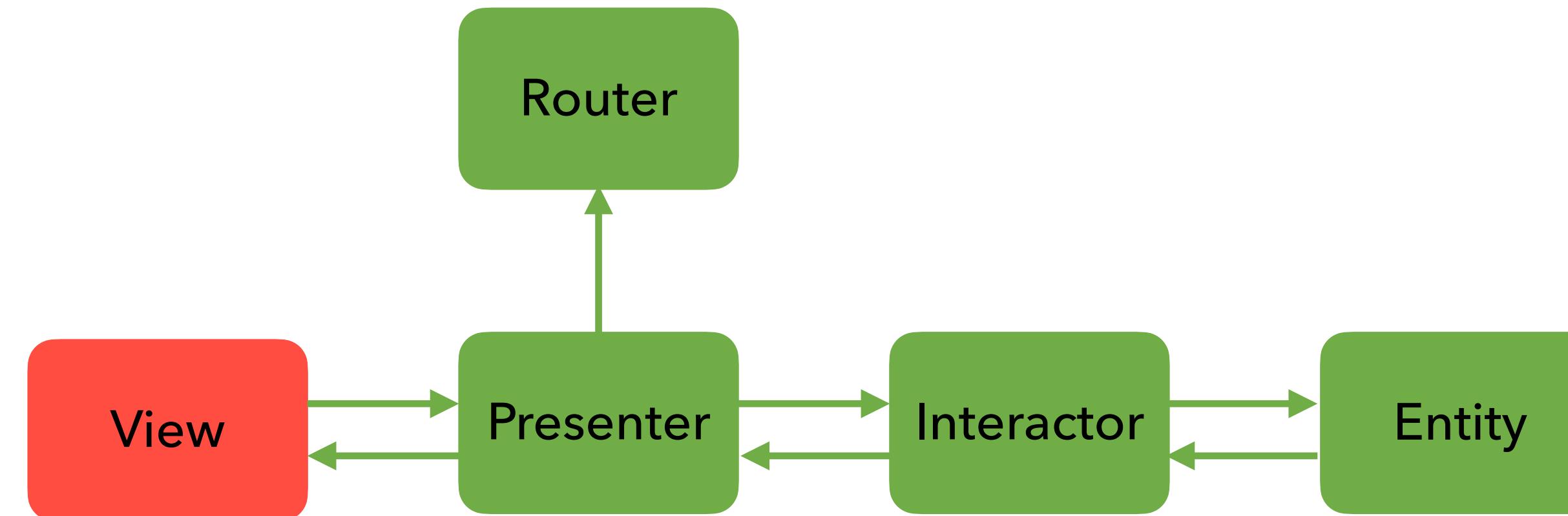


```
1 func forceLogging(enable: Bool?) {  
2     guard let enable else {  
3         viewDispatch.async { $0.showAlertFailedSaving(error: "") }  
4         return  
5     }  
6     viewDispatch.async { $0.showAlertForceLogging(enable: enable) }  
7 }
```

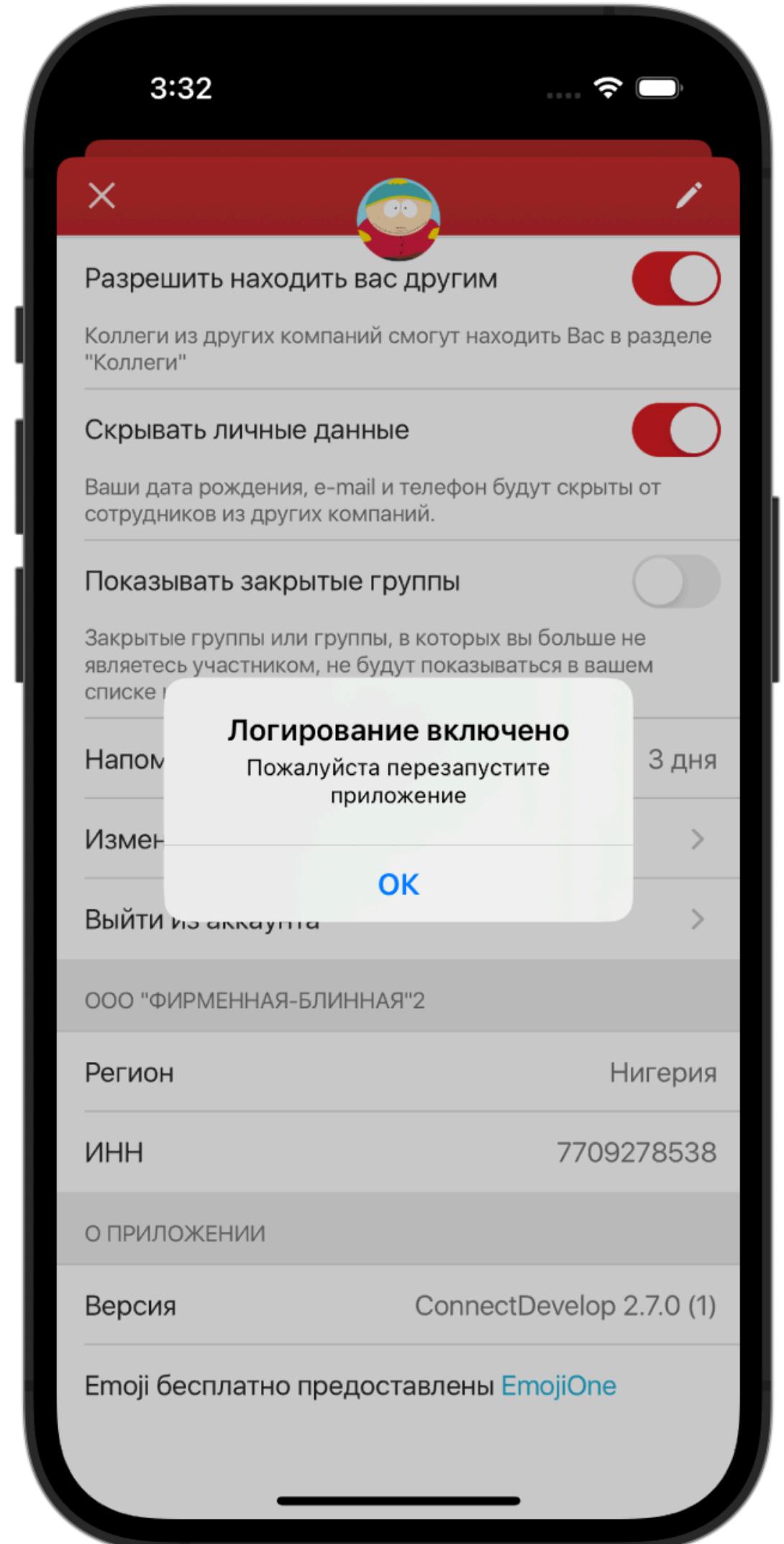


# Под капотом

```
1 func showAlertForceLogging(enable: Bool) {  
2     let alertController = UIAlertController(  
3         title: enable ? "Logging enabled".localized() : "Logging disabled".localized(),  
4         message: "Please restart the application".localized(),  
5         preferredStyle: .alert  
6     )  
7     let continueAction = UIAlertAction(title: "OK".localized(), style: .cancel)  
8     alertController.addAction(continueAction)  
9  
10    present(alertController, animated: true, completion: nil)  
11 }
```



# Под капотом



# Плюсы и минусы

Приложение разделено на слои

# Плюсы и минусы

Есть возможность  
протестировать каждый слой

# Плюсы и минусы

у каждой команды / разработчика  
свое виденье архитектуры VIPER

# Плюсы и минусы

VIPER не спасает от плохого кода

# Плюсы и минусы

Сложности с очередями при наследовании классов если каждый слой работает в своей очереди

Два стула:

- 1) Создание отдельной очереди для подкласса.
- 2) Создание пустых методов в родительском классе и их переопределение в дочернем классе.

# Плюсы и минусы

Сложности с очередями при наследовании классов если каждый слой работает в своей очереди

Два стула:

- 1) Создание отдельной очереди для подкласса.
- 2) Создание пустых методов в родительском классе и их переопределение в дочернем классе.

# Плюсы и минусы

Сложности с очередями при наследовании классов если каждый слой работает в своей очереди

Два стула:

- 1) Создание отдельной очереди для подкласса.
- 2) Создание пустых методов в родительском классе и их переопределение в дочернем классе.

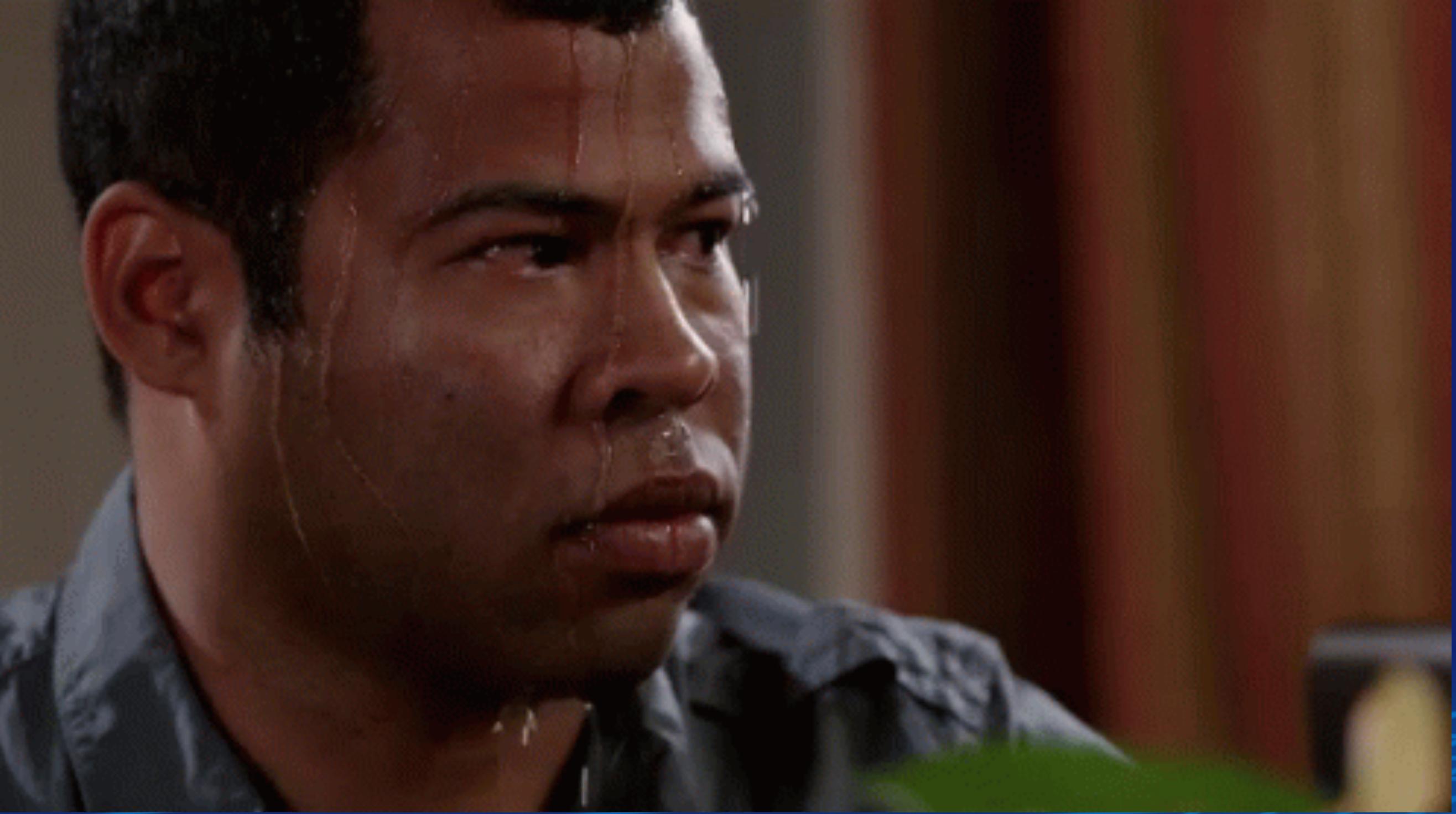
# Вопросы?

Опыт разработки iOS  
Применение VIPER



Сергей Долгих  
iOS разработчик,  
1С-Коннект

# Вопросы?



Опыт разработки iOS  
Применение VIPER



Сергей Долгих  
iOS разработчик,  
1C-Коннект