

Яндекс

Архитектурный подход к обработке ошибок

Дмитрий Михайлов

Яндекс Толока

Заработок в интернете



Выбирайте задания

☰ Все задания

★★★★★
Какое у кота настроение?
Посмотрите на картинку и определите настроение кота.
0,04\$ за задание


★★★★★
Обновление данных об организациях
Пройдите адреса цветочных магазинов и сфотографируйте места, режим работы и другую информацию.
0,40–0,80\$ за задание

★★★★★ полевые задания · 18+
Модерация коротких видео
Посмотрите видео и скажите, к какой

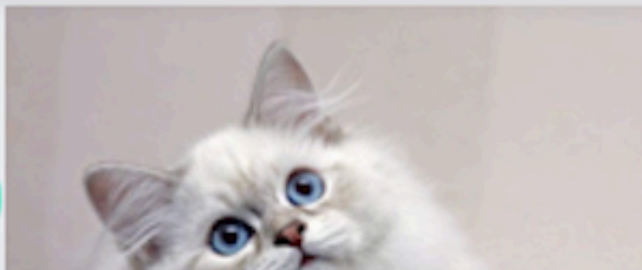
Выполняйте дома

← 23:34:08 | 0,20 \$

Какое у кота настроение?



Хорошее Плохое



... или на прогулке

0,04–0,80\$ 9

0,70\$ 3

0,07–0,50\$ 17

0,10–0,20\$ 6

0,07–0,50\$ 23

Получайте вознаграждение

☰ Мои деньги: 1,50 \$

Обновление данных об организациях 1,60 \$

30 ноября	1,66 \$
Бок-о-бок, поиск 9 заданий Я.Единогор	0,18 \$
Бок-о-бок, мобильные сайты 9 заданий Я.Единогор	0,18 \$
Фасады организаций 8 заданий Я.Единогор	0,80 \$
Время работы организаций 5 заданий Я.Единогор	0,50 \$
18 мая 2018	3,72 \$
Бонус за регистрацию МММ или ССР	0,01 \$

Только две вещи бесконечны -
Вселенная и человеческая
глупость.

Хотя насчет Вселенной я не
уверен.

Альберт Эйнштейн

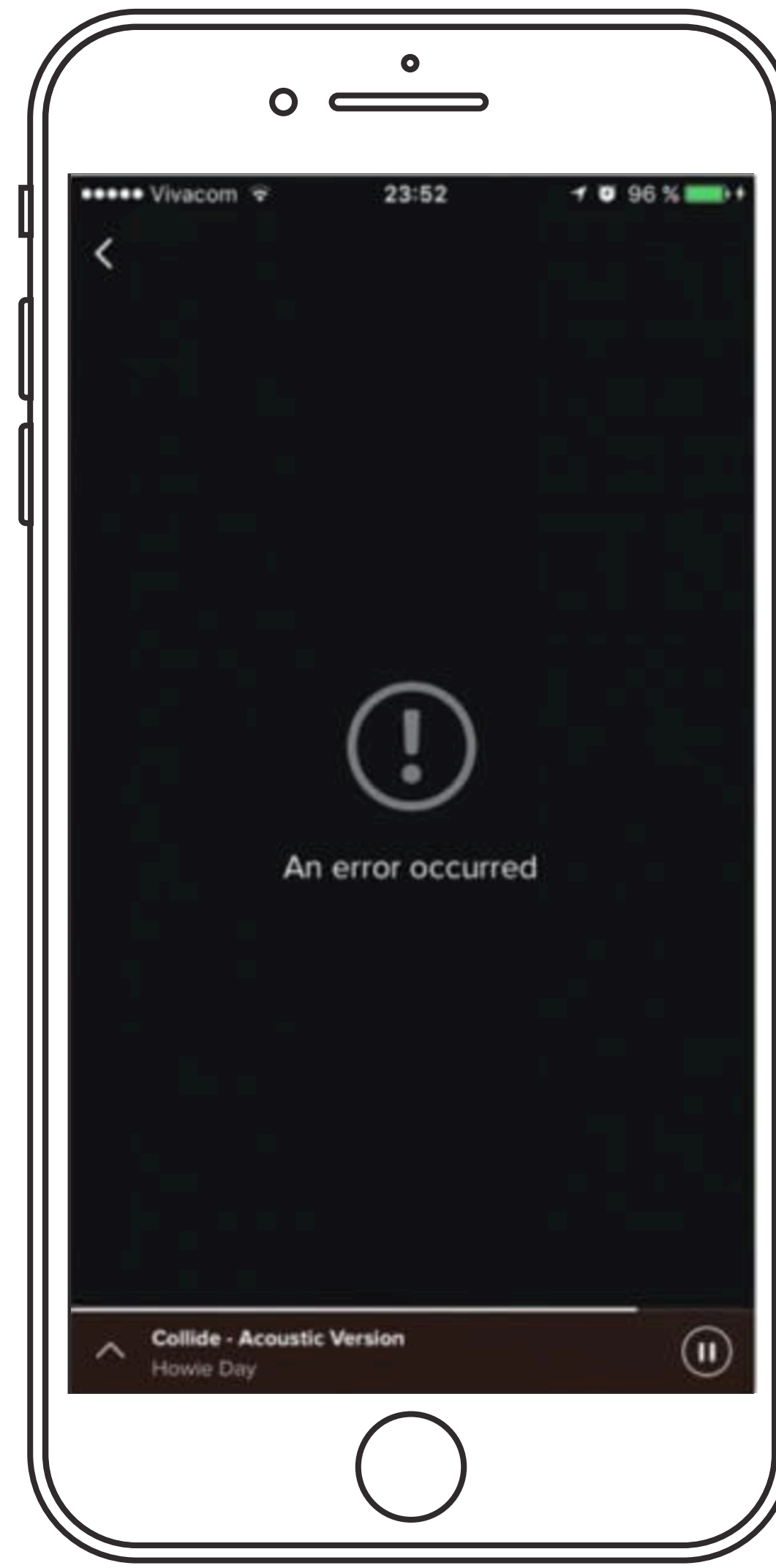
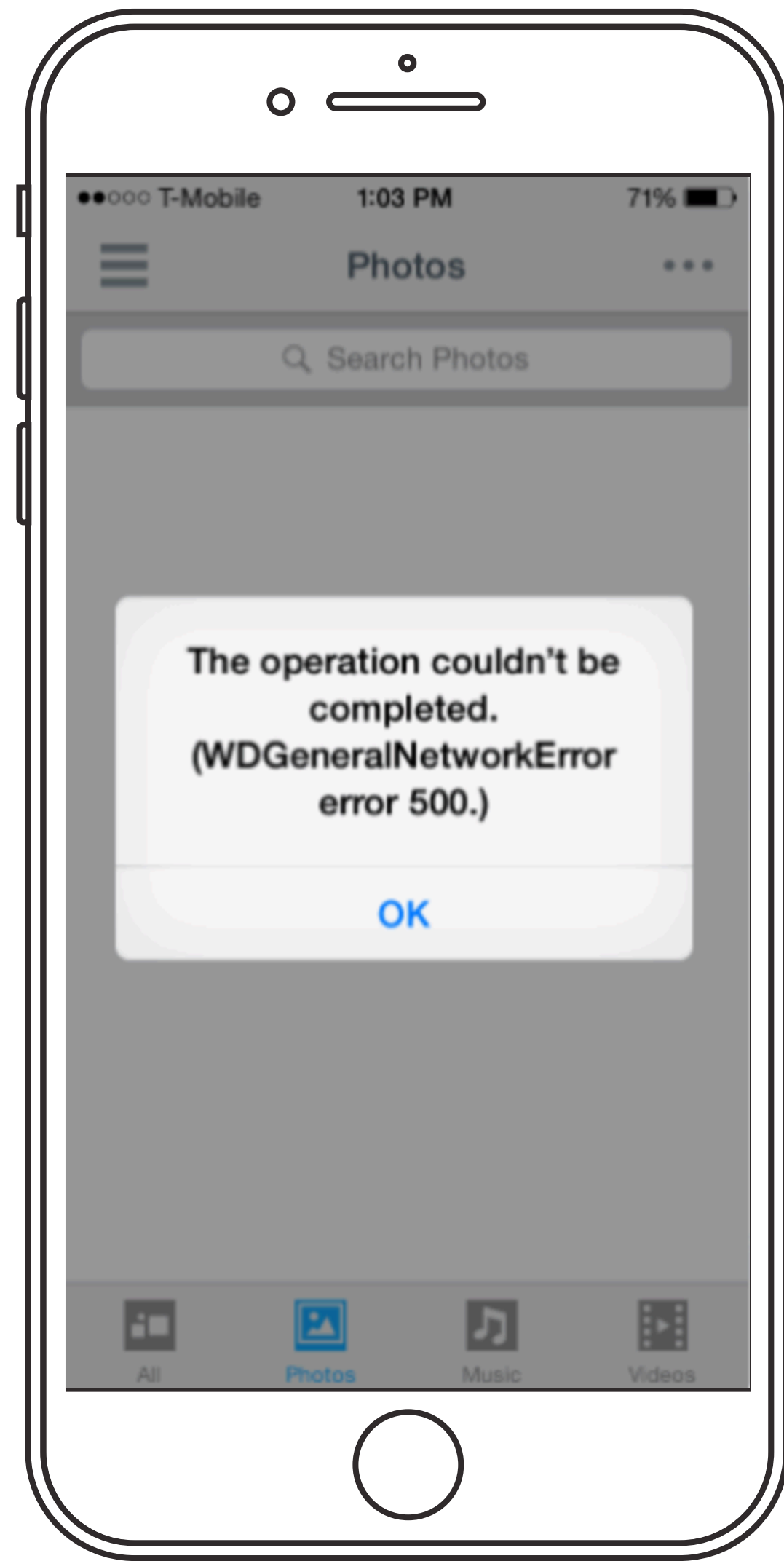
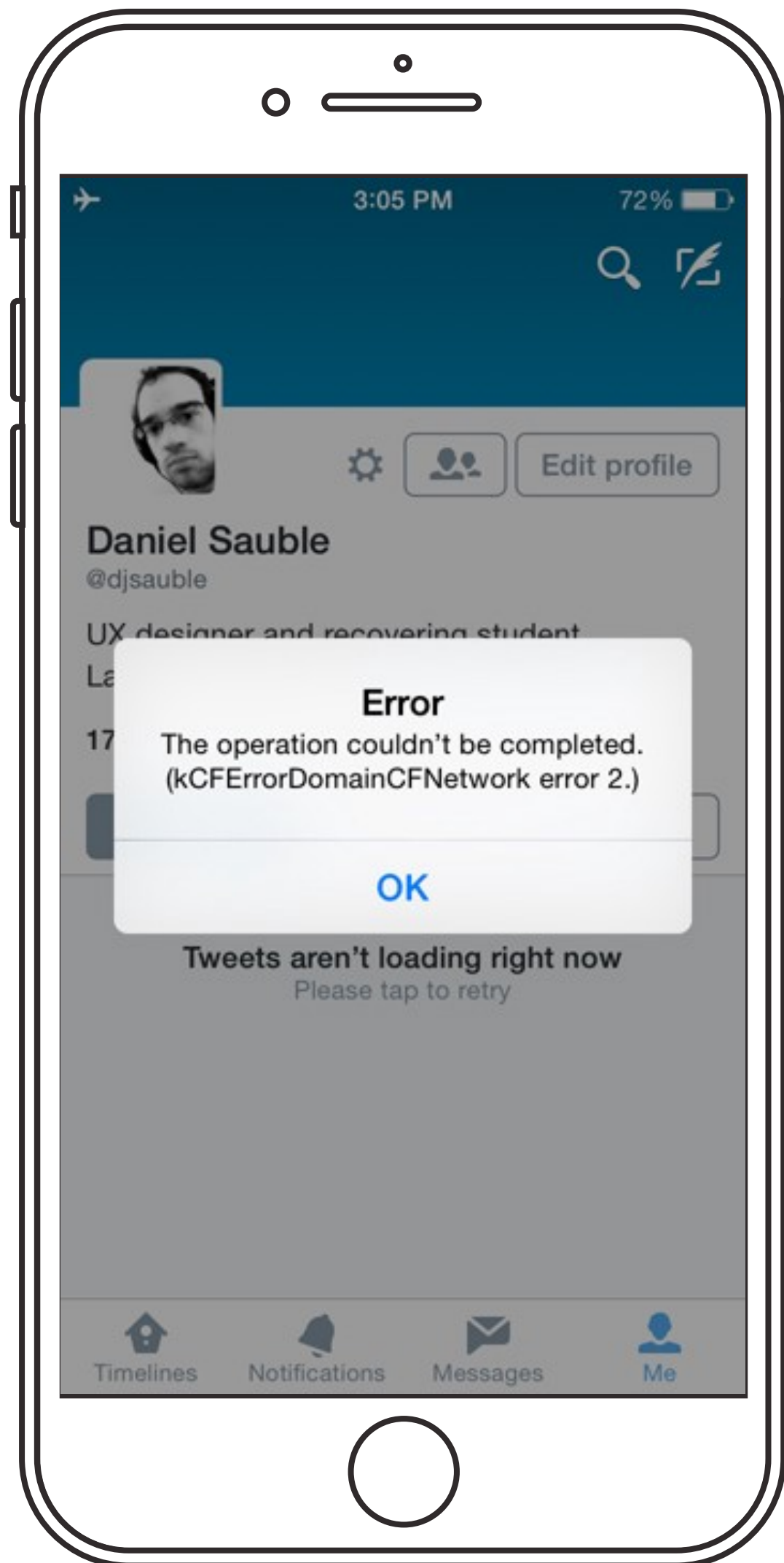
Ошибки неизбежны

- › Баги в нашем коде
- › Баги в библиотеках или в iOS
- › Ошибки бэкенда
- › Сбои сетевого соединения
- › Ошибки пользователей

Показатели хорошей обработки ошибок

- › Легко для разработчика: единообразно и безопасно
- › Безболезненно для пользователя: понятные и восстанавливаемые
- › Эффективно для команды: удобный мониторинг и анализ
- › Полезно для проекта: меньше ошибок, короткие циклы багфиксов

Плохая обработка ошибок



- › Вызывает недоумение пользователей
- › Теряется ценная информация об ошибочной ситуации
- › Бессистемность обработки

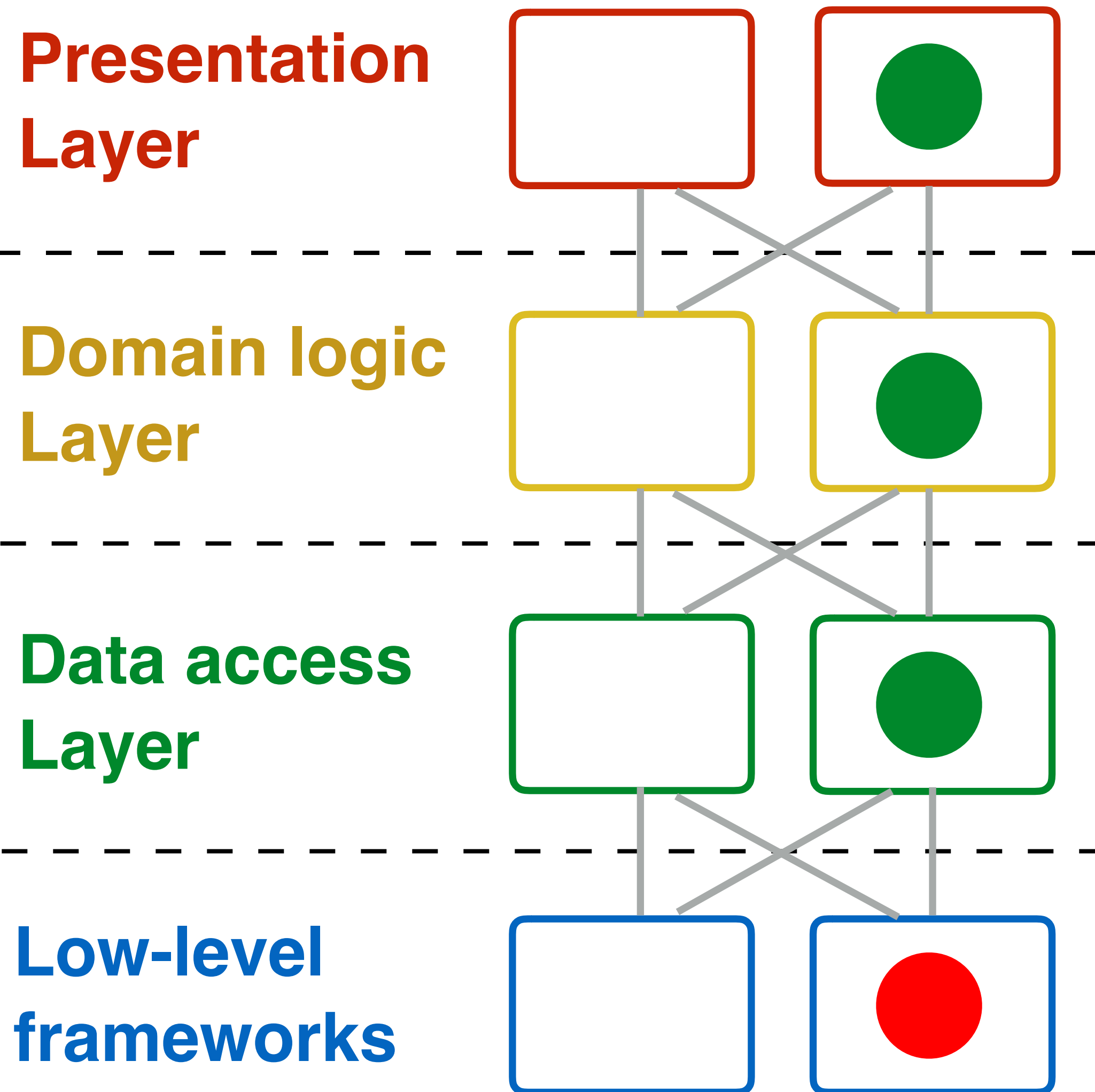
Результаты плохой обработки ошибок

- › Уменьшается желание пользователей работать с приложением, падает retention
- › Повышается нагрузка на службу поддержки, падает рейтинг, число установок
- › Усложняется поиск причин ошибок, удлиняется цикл багфикса
- › Ухудшается качество кода, фрагментируется архитектура
- › Разработка и поддержка становятся дороже

Дело снова в архитектуре



Обработка ошибок - архитектурный аспект

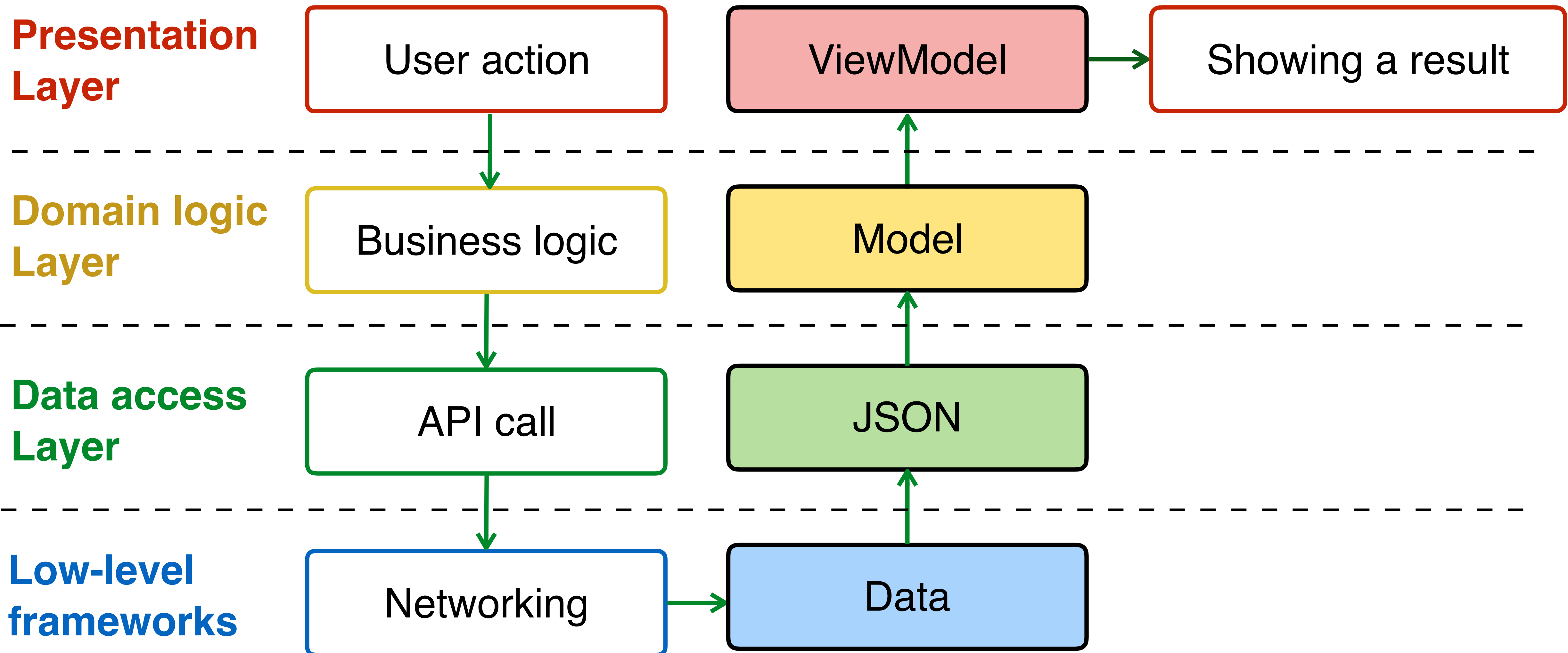


- › Где может возникнуть ошибка?
- › Где обрабатывать ошибку?
- › Как “осознать” ошибочную ситуацию в месте её обработки?

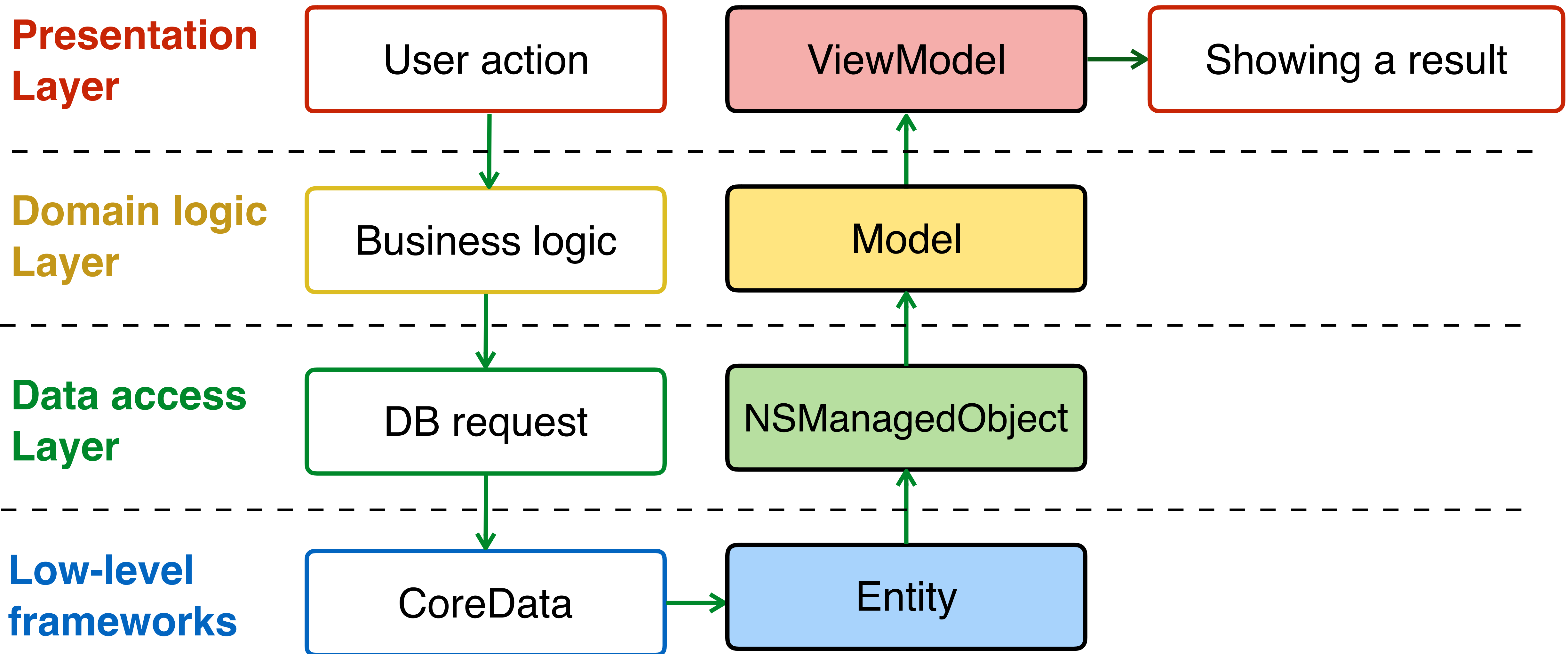
Как помочь обработчику “осознать” ошибку

- › Ошибка – ценный результат действия, который нельзя терять
- › Причина+контекст ошибки на каждом из слоев
- › Ошибочная ситуация – не одно событие, а цепь событий – попыток выполнить операцию каждым уровнем
- › Такую цепь можно представить в виде связанного списка объектов-ошибок
- › У каждого слоя/модуля – свои виды и коды ошибок, соответствующие его уровню компетенции

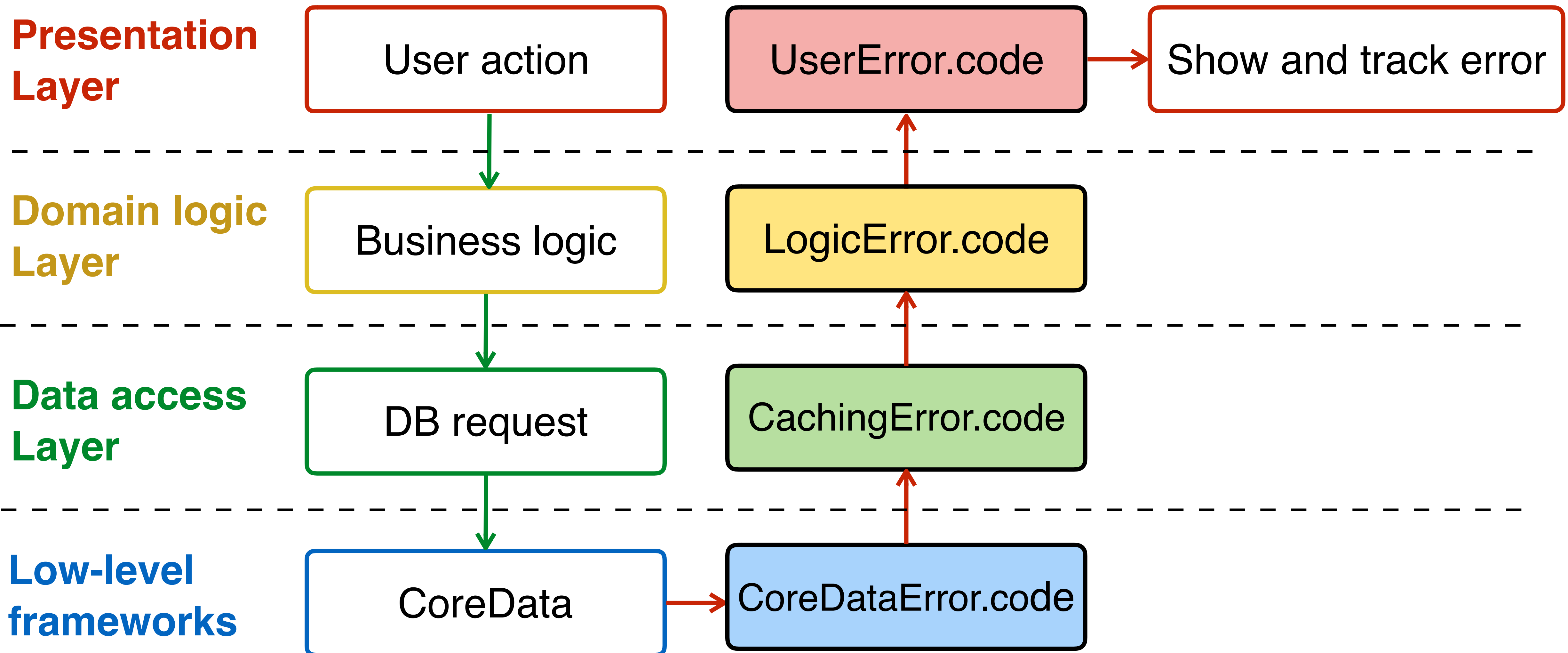
Передача действия — возврат данных



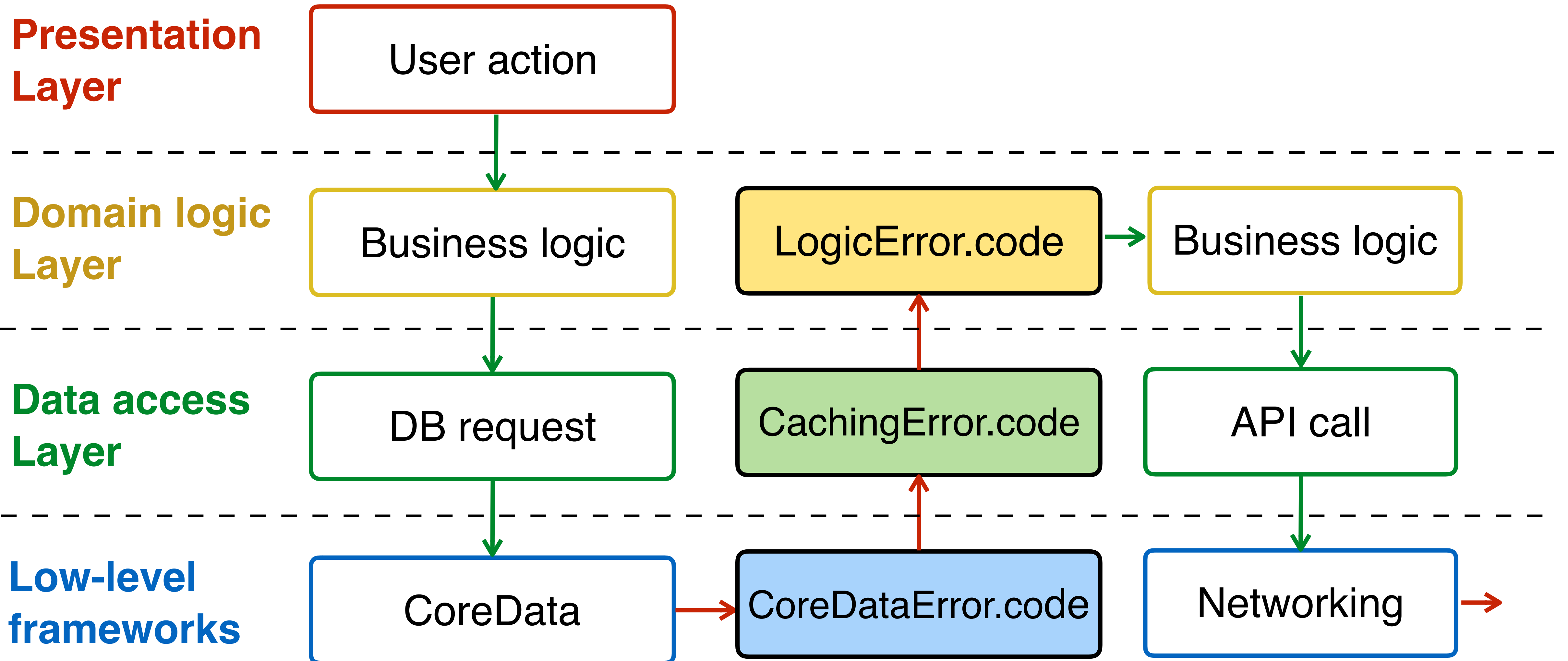
Передача действия — возврат данных



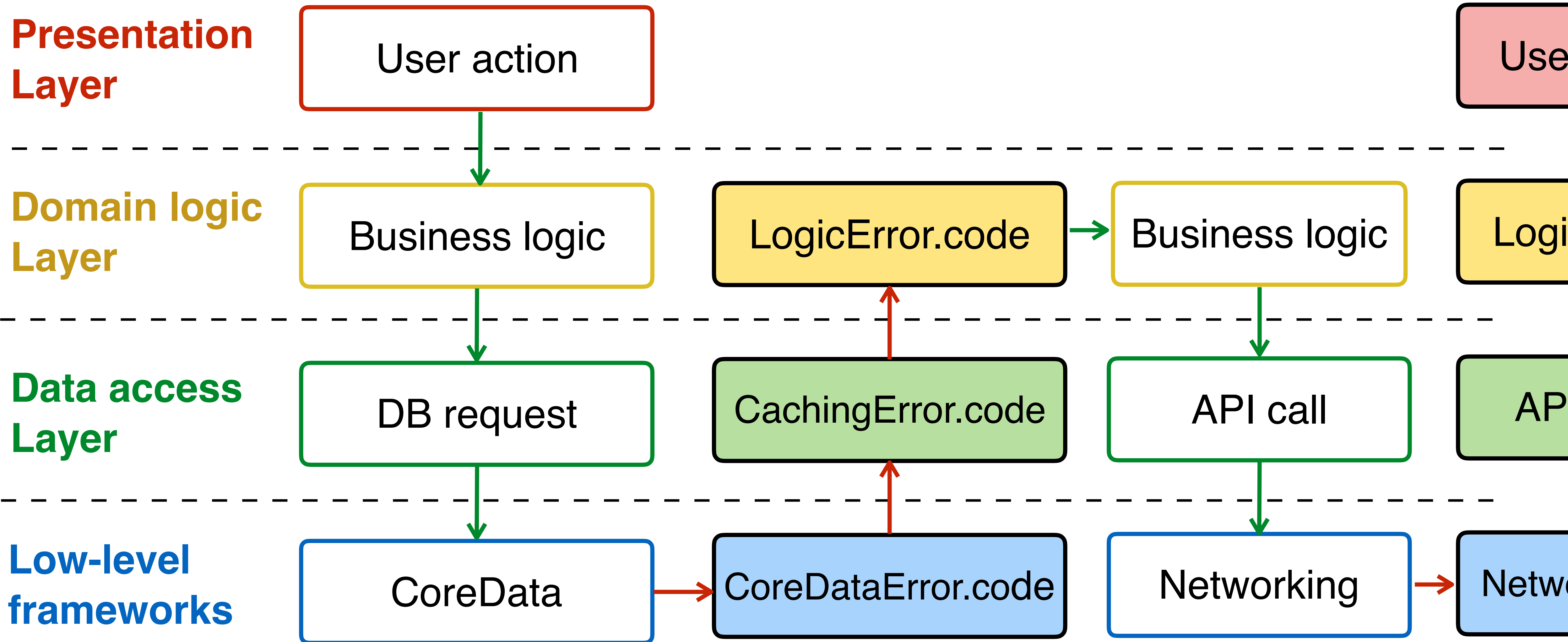
Передача действия — возврат ошибки



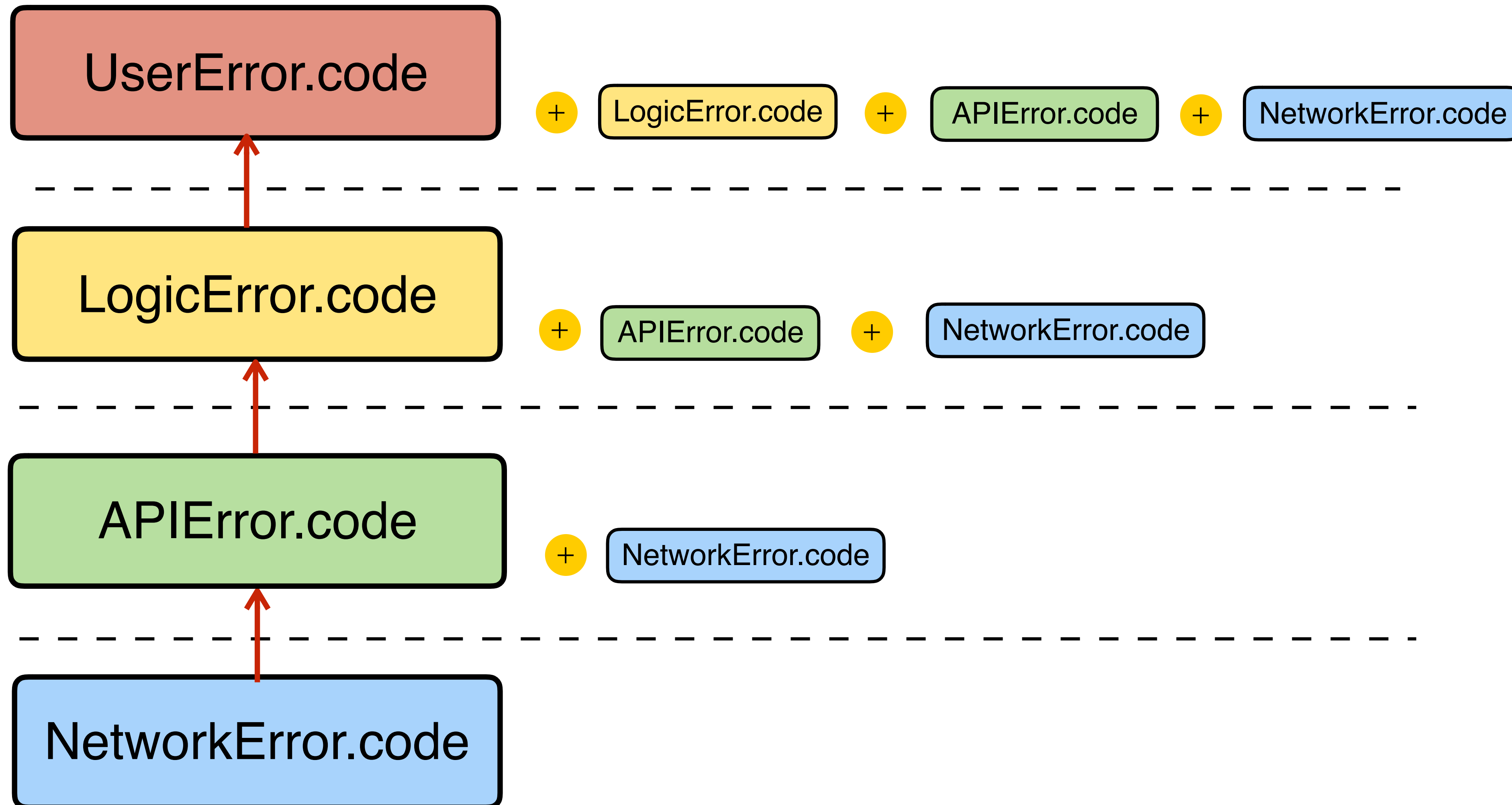
Передача действия — возврат ошибки



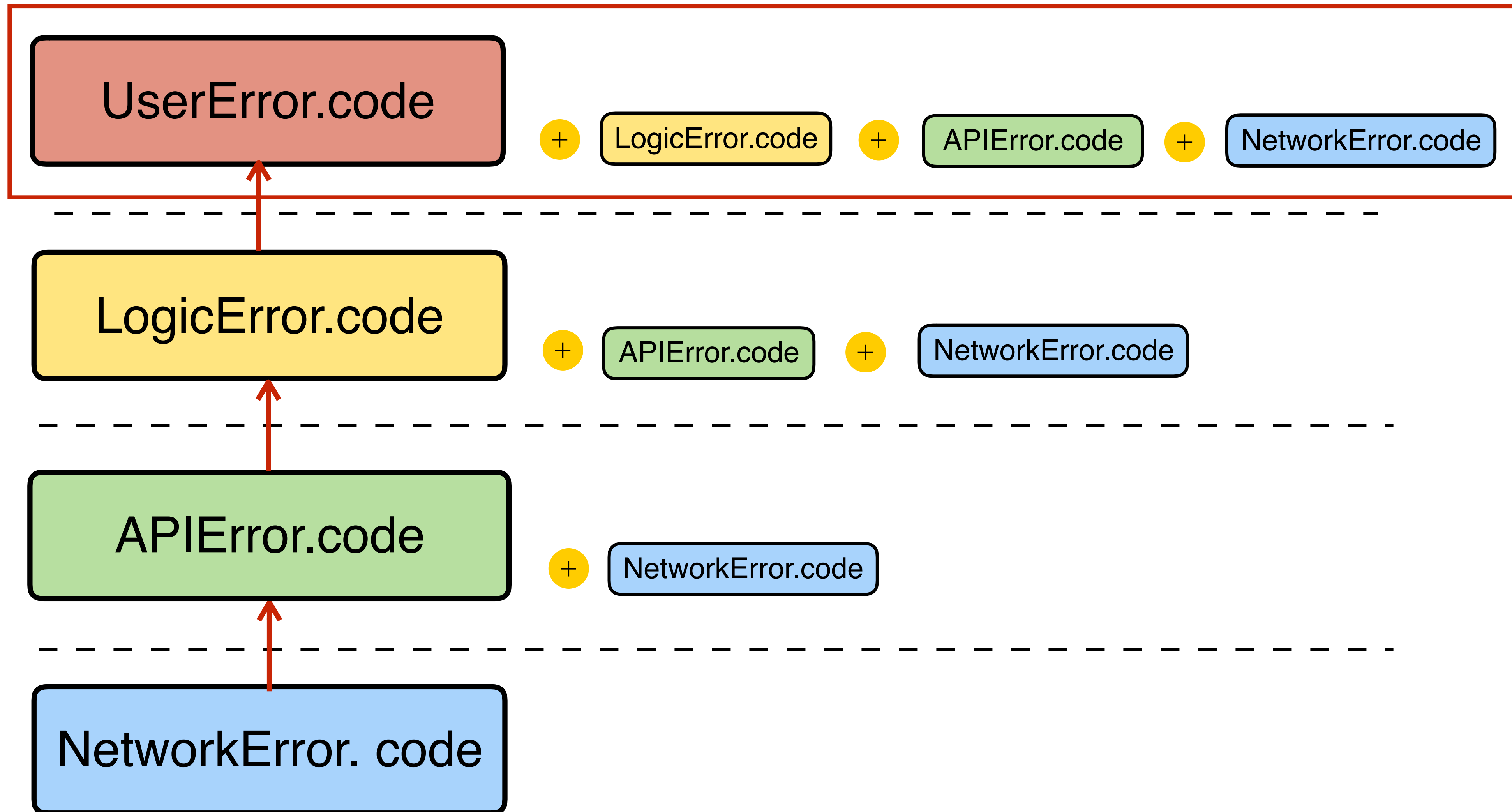
Передача действия — возврат ошибки



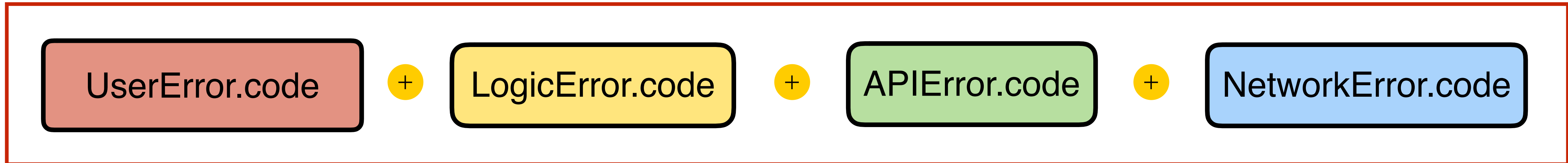
Создание цепочки для ошибочной ситуации



Создание цепочки для ошибочной ситуации



Цепочка ошибок для ошибочной ситуации



- › Низлежащая ошибка — причина вышестоящей, Вышестоящая ошибка — контекст для низлежащей
- › Такая цепочка уникальна и стабильна для отслеживания
- › Представление для идентификации: короткое и полное

US14-LO43-AP7-NE8

UserError.connectionIssue - LogicError.loginFailed - APIError.getTokenFailed - NetworkError.timeout

Итого: архитектурные решения

- › Разделение всех ошибок по уровням компетенции
- › У каждого слоя/модуля – свои виды и коды ошибок
- › При возврате (всплытии) ошибки – оборачиваем ее в другую ошибку на каждом слое
- › Создание цепочки событий, которая описывает весь путь, но без ненужных деталей реализации

Погружаемся в код



Из документации Apple...

```
enum IntParsingError: Error {  
    case overflow  
    case invalidInput(String)  
}
```

```
extension Int {  
    init(validating input: String) throws {  
        if !_isValid(s) { throw IntParsingError.invalidInput(s) }  
    }  
}
```

```
do {  
    let price = try Int(validating: "$100")  
} catch IntParsingError.invalidInput(let invalid) {  
    print("Invalid character: '\(invalid)'")  
} catch IntParsingError.overflow {  
    print("Overflow error")  
} catch {  
    print("Other error")  
}
```

Из документации Apple...

```
struct XMLParsingError: Error {
    enum ErrorKind {
        case invalidCharacter
        case mismatchedTag
    }
    let line: Int
    let column: Int
    let kind: ErrorKind
}
```

```
func parse(_ source: String) throws -> XMLDoc {
    throw XMLParsingError(line: 19, column: 5, kind: .mismatchedTag)
}
```

```
do {
    let xmlDoc = try parse(myXMLData)
} catch let e as XMLParsingError {
    print("Parsing error: \(e.kind) [\(e.line):\(e.column)]")
} catch {
    print("Other error: \(error)")
}
```

Базовый класс для иерархии ошибок

- › Строгая типизация подклассами ошибок, работа с цепочками
- › Выбор кандидата на базовый класс:
 - › NSError – слабая типизация (строка + число + словарь).
Плохие образцы для подражания – в NSCocoaErrorDomain
 - › Протокол Error (+ протоколы LocalizedError, RecoverableError, CustomNSError...)
 - › Собственный класс

Базовый класс ошибки

```
public protocol BaseErrorCode: RawRepresentable where RawValue == Int {}
```

```
open class BaseError<ErrorCode: BaseErrorCode>: NSError {  
    //or implement LocalizedError, RecoverableError, CustomNSError protocols for Error
```

```
public required init(code: ErrorCode, underlying: Error? = nil, systemMsg: String? = nil) {  
    let userInfo = [NSUnderlyingErrorKey: underlying, NSFailureReasonErrorKey: systemMsg]  
    super.init(domain: domain, code: code.rawValue, userInfo: userInfo)  
}
```

```
public var domain: String {  
    return String(describing: type(of: self))  
}  
public var errorCode: ErrorCode {  
    return ErrorCode(rawValue: code)!  
}  
public var errorCodeName: String {  
    return String(describing: errorCode)  
}
```

```
...
```

```
}
```


Дочерние классы ошибок

```
public enum GeoErrorCode: Int, BaseErrorCode {  
    case undefined = 0  
    case locationServicesDisabled = 1  
    case notAuthorizedYet = 2  
    case authorizationDenied = 3  
    case locationCannotBeDetermined = 4  
    case locationAccuracyIsNotEnough = 5  
}
```

```
public class GeoError: BaseError <GeoErrorCode> {  
    public override func domainShortname() -> String {  
        return "GE"  
    }  
}
```

```
...  
let error = GeoError(code: .locationAccuracyIsNotEnough)
```

```
...  
track(error)  
show(error)
```

Возврат ошибок (пример из бизнес-слоя)

```
// Business-logic layer (wraps errors into InteractorError type)
public class EnvironmentInteractor: EnvironmentInteractorProtocol {
    public func checkAppVersion(success: @escaping (Bool) -> Void,
                                failure: @escaping (InteractorError) -> Void) {
        environmentAPIDataManager.platformInfo(success: { platformInfo in
            ...
        }, failure: { error in //APIError
            failure(InteractorError(code: .checkAppVersion, underlying: error))
        })
    }
}
```

Возврат ошибок (пример из UX-слоя)

```
// Presentation layer (wraps errors into UserError type)
public class AuthPresenter: AuthPresenterProtocol {
    public func viewDidAppear() {
        environmentInteractor.checkAppVersion(success: { isValid in
            ...
        }, failure: { error in //InteractorError
            showAndTrack(UserError(with: error))
        })
    }
}
```

Recovery handlers (обработка внутри слоя)

```
public protocol RecoverableErrorHandlerProtocol {  
    func recover(_ error: BaseError,  
                currentRetryAttempt: Int,  
                success: @escaping (ErrorRecoveryResult) -> Void,  
                failure: @escaping BaseErrorClosure)  
}
```

```
public enum ErrorRecoveryResult: Equatable {  
    case cannotHandle //this handler is not intended to recover such kind of error  
    case didHandle    //recovery was potentially successful and action can be retired  
}
```

Recovery handlers (пример из слоя API)

```
func tolokaAPIRecoverableErrorsHandlers() -> [RecoverableErrorHandlerProtocol] {  
    let backendTempErrorsHandler: BackendTempErrorsHandlerProtocol = resolve()  
    let authErrorHandler: ForbiddenErrorHandlerProtocol = resolve()  
    let captchaRequiredErrorHandler: CaptchaRequiredErrorHandlerProtocol = resolve()  
  
    return [backendTempErrorsHandler, authErrorHandler, captchaRequiredErrorHandler]  
}  
  
public class APIDataManager {  
    func request(...) {  
        backendDataSource.request(..., success: ..., failure: { error in  
            let recoverers = tolokaAPIRecoverableErrorsHandlers()  
            //recursively calling recover(error, success: ..., failure: ...) for all handlers  
            //until we get .didHandle and then we repeat request(...)  
        })  
    }  
}
```

Итого: аспекты реализации

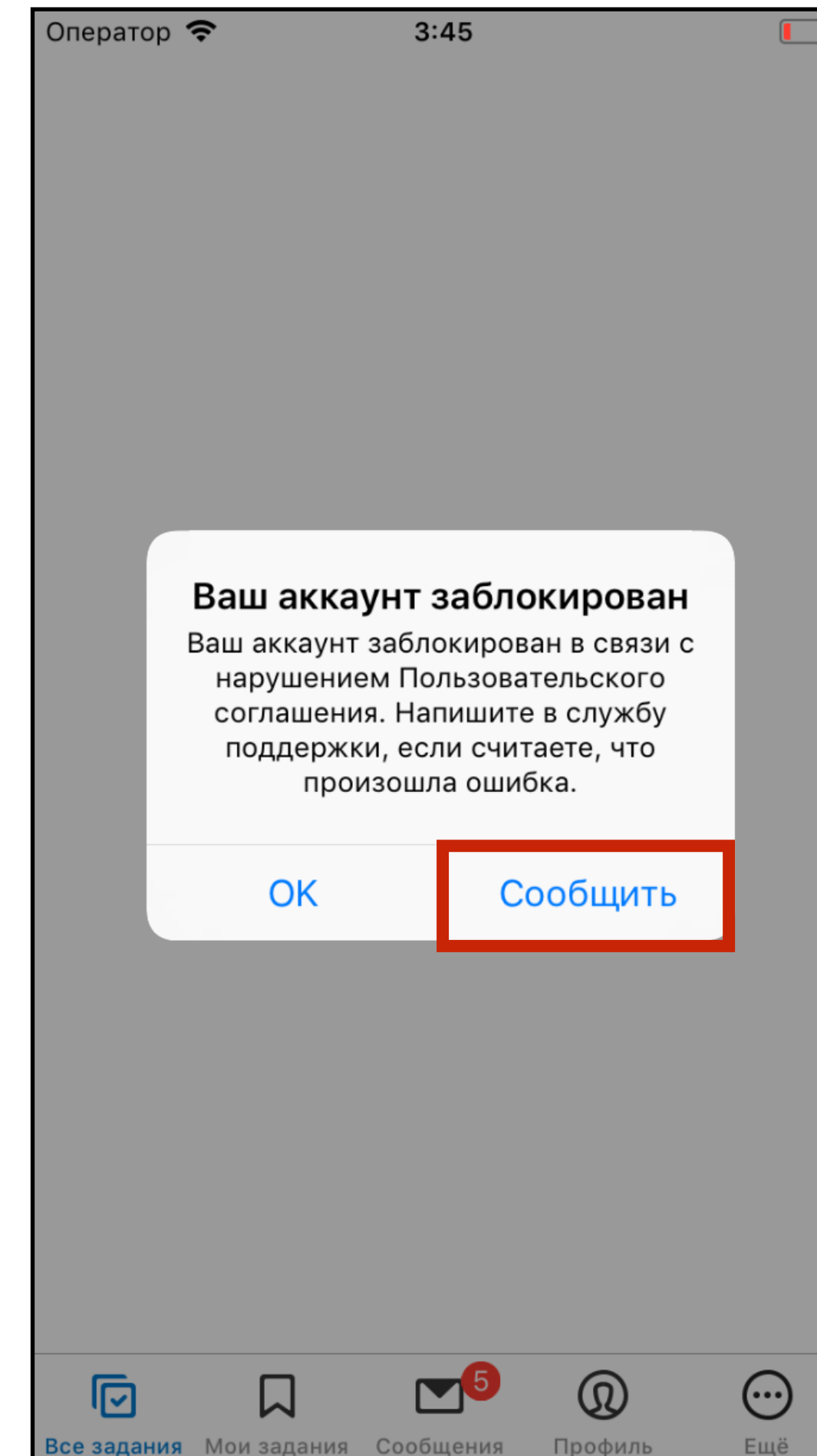
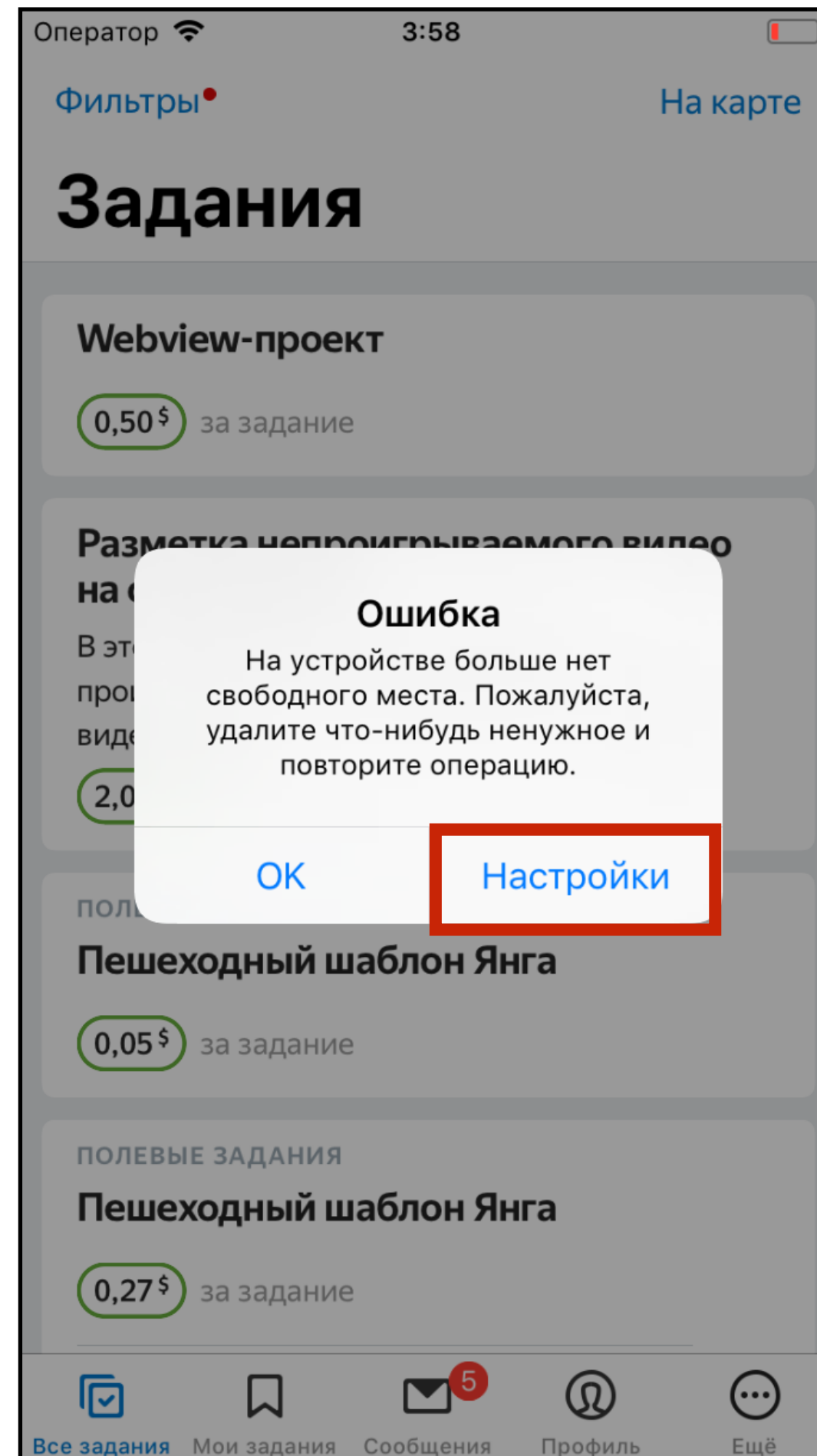
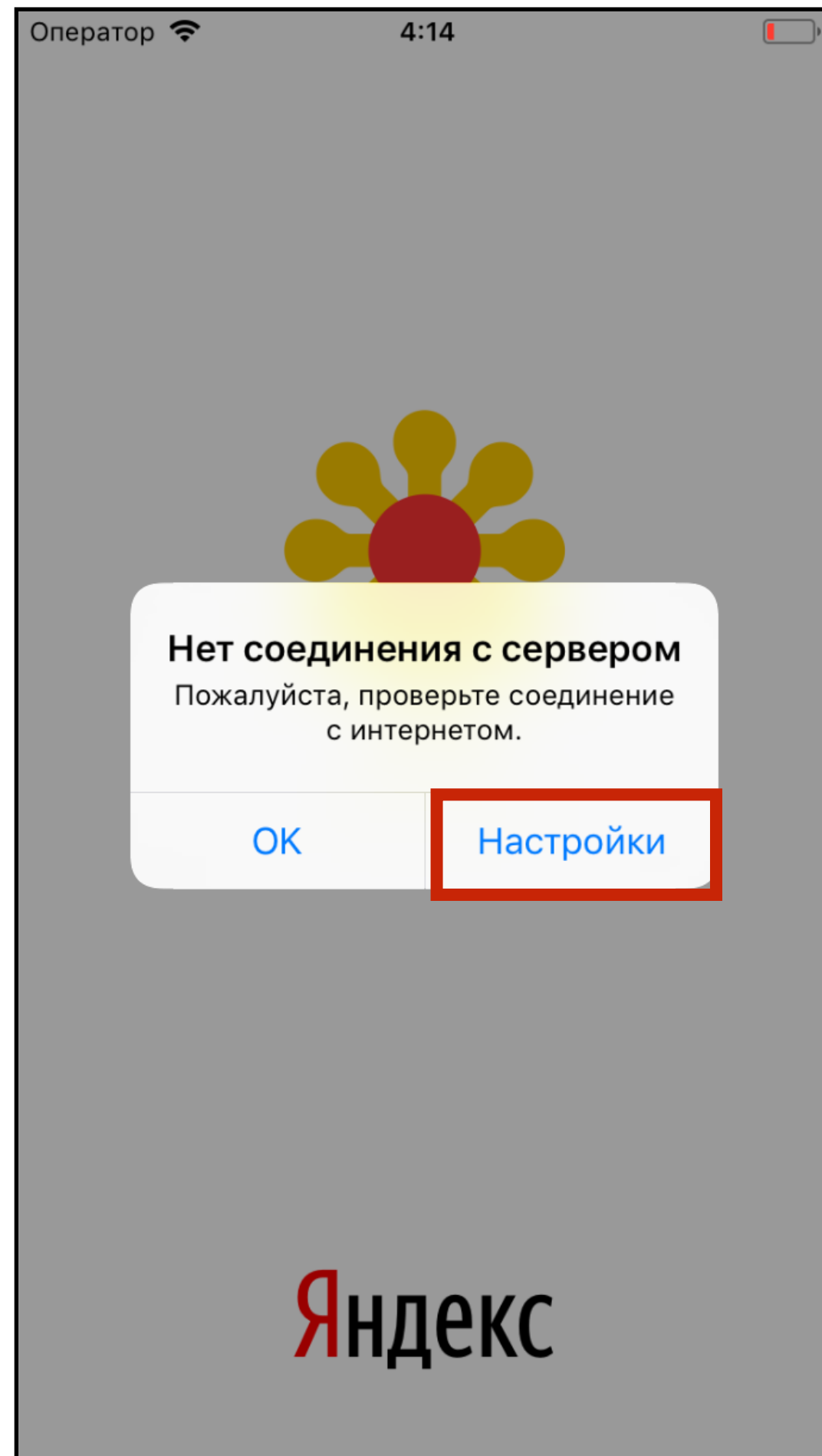
- › Максимум помощи от компилятора: типизируем возвращаемые ошибки
- › Слоям/модулям - свои подтипы ошибок
- › Ограничиваем использование do-catch – только локально внутри модулей
- › Базовый класс, умеющий работать с цепочками ошибок
- › Обобщаем подход обработке и восстановлению от ошибок

Снова к пользователям



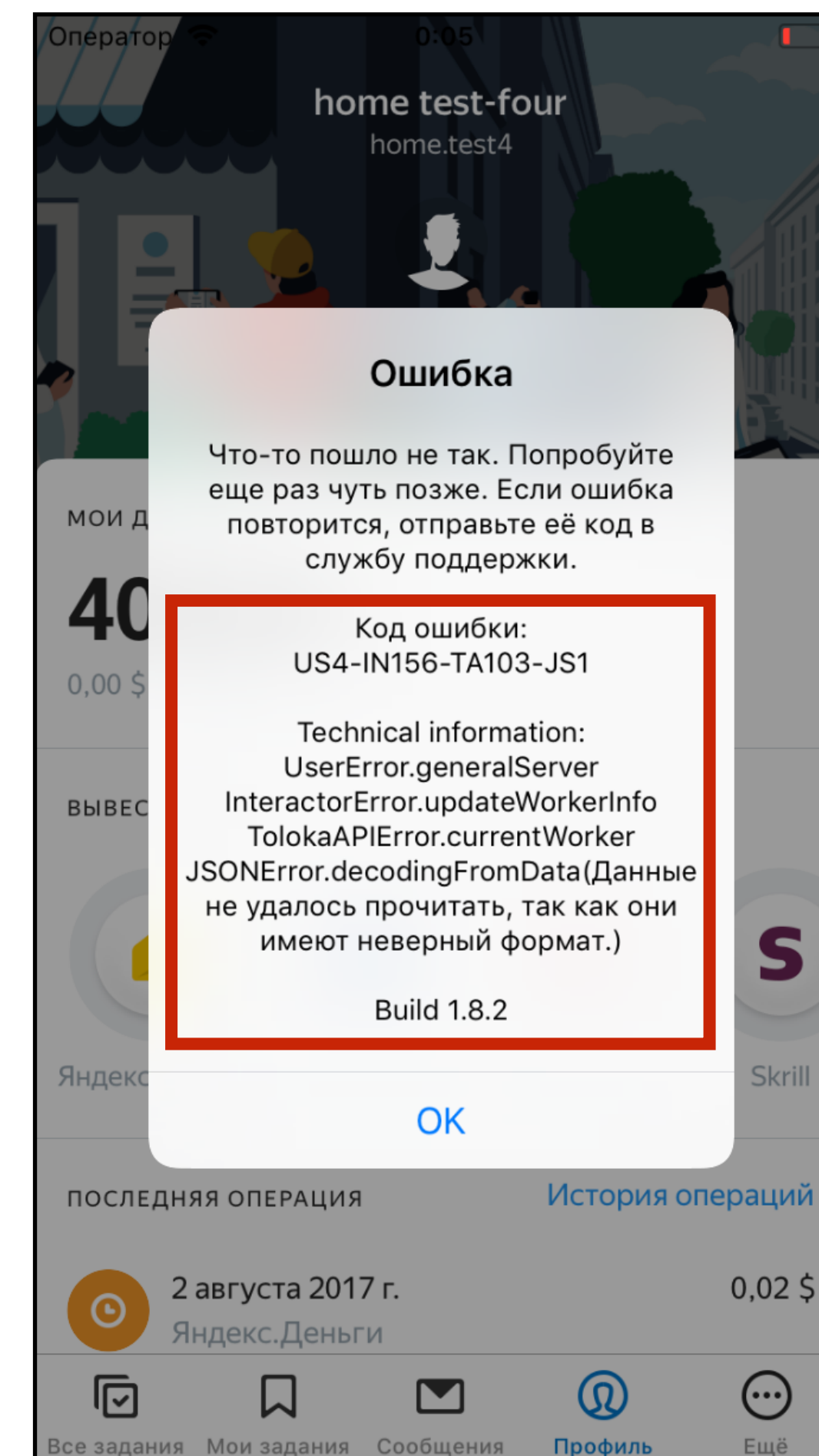
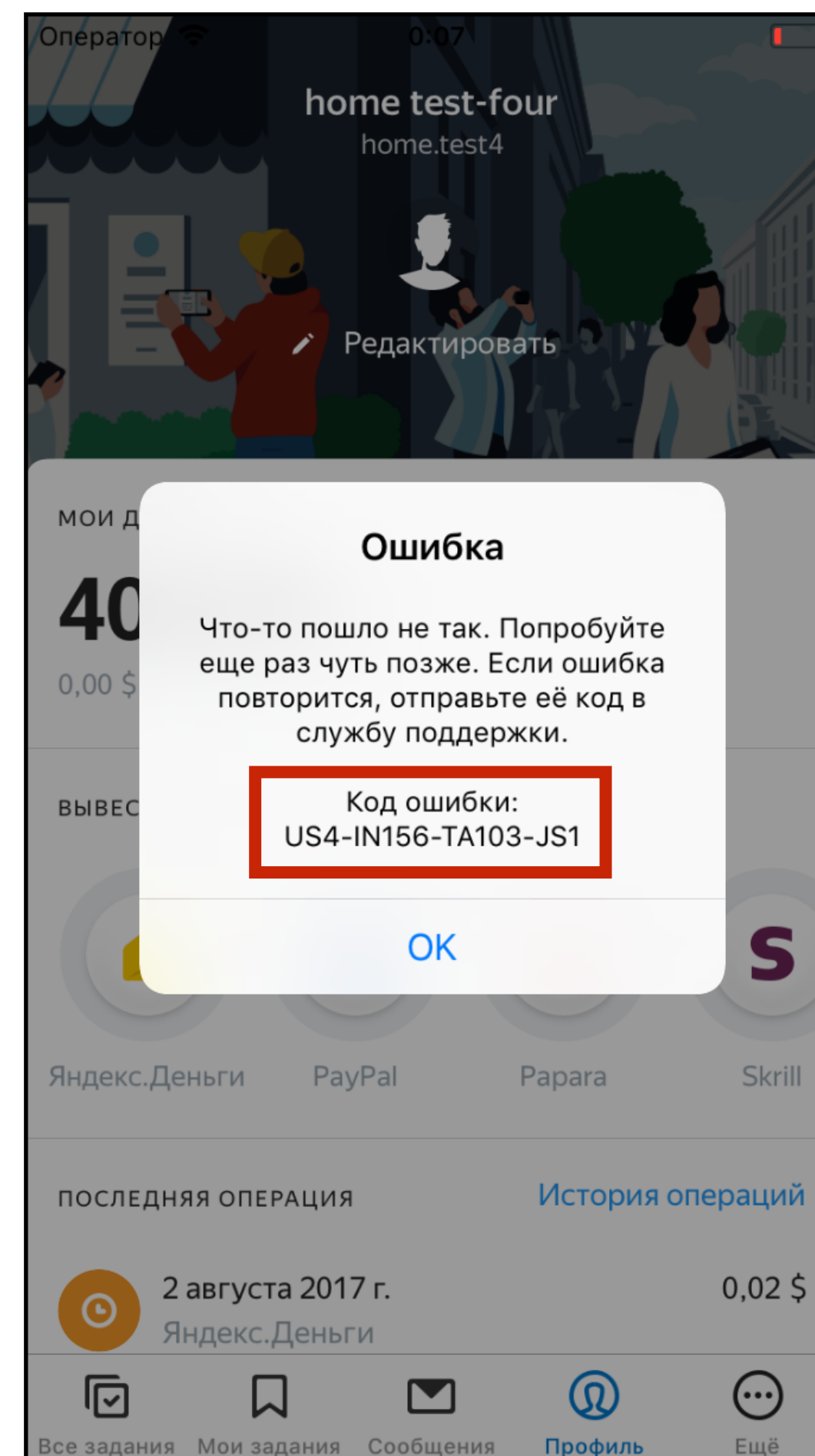
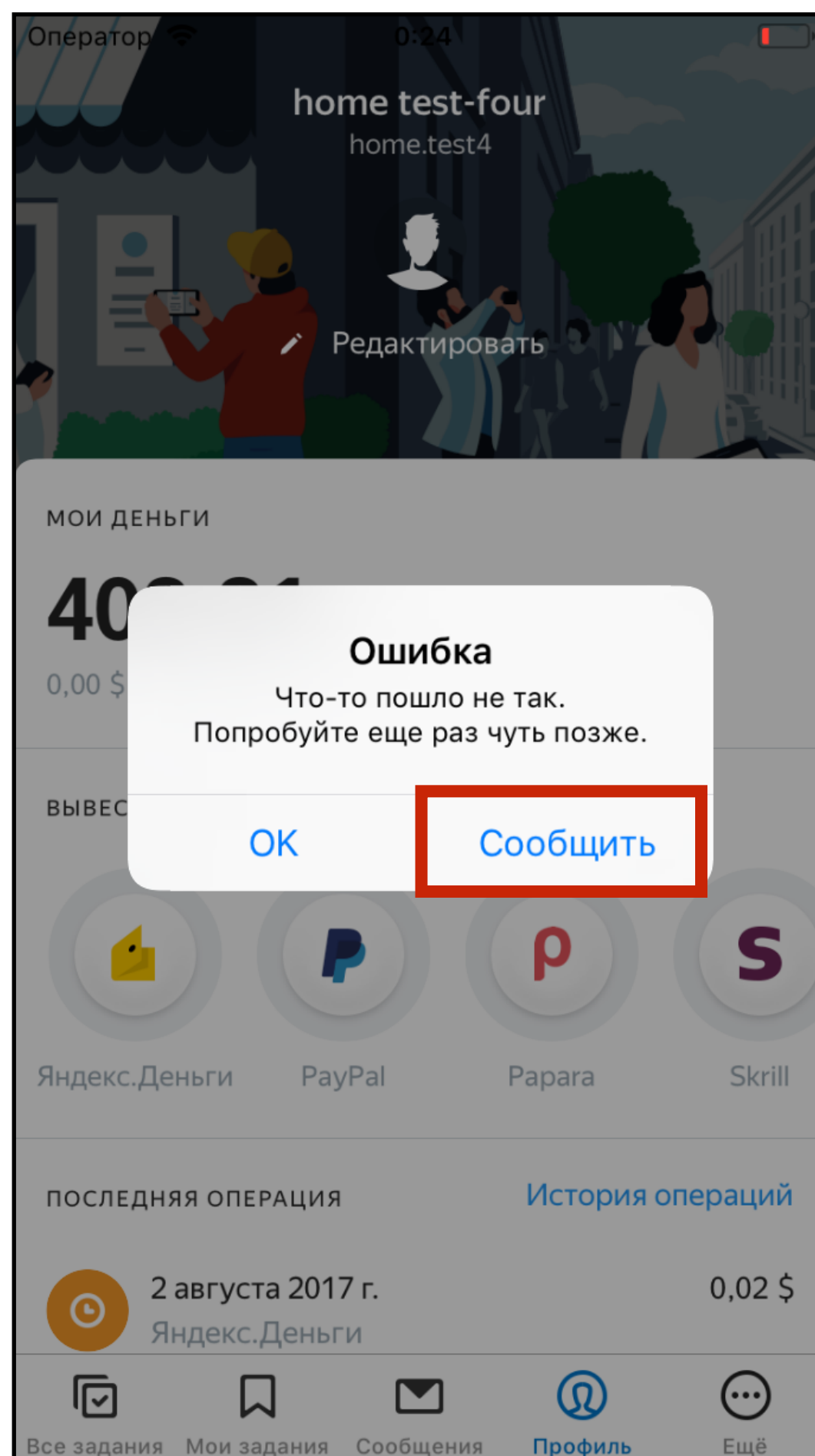
Восстановимые ошибки

Информативность (что это было?) и полезность (что теперь



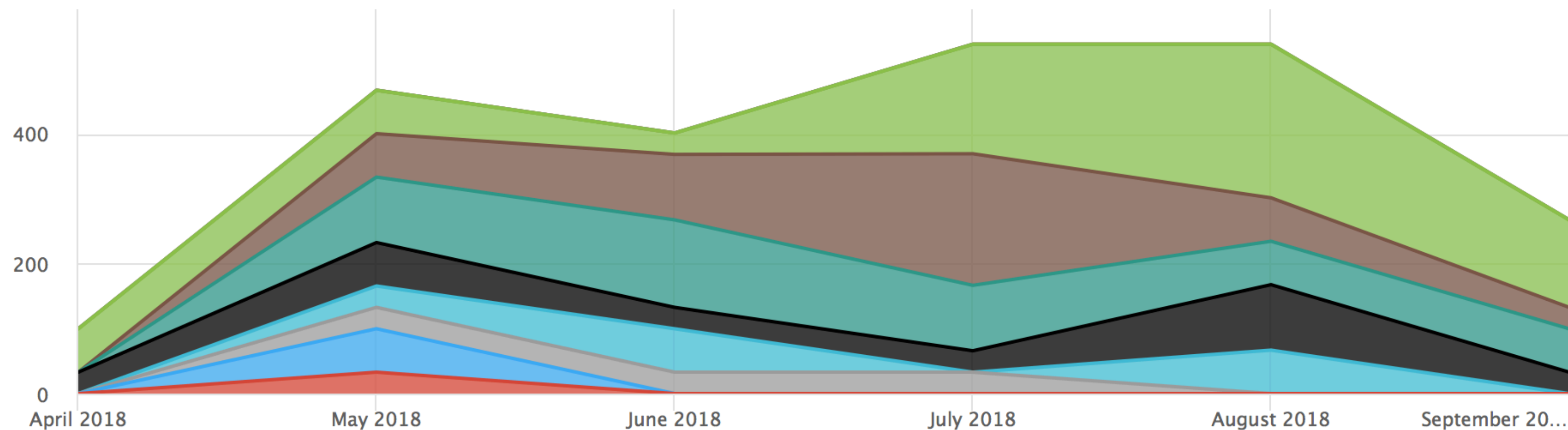
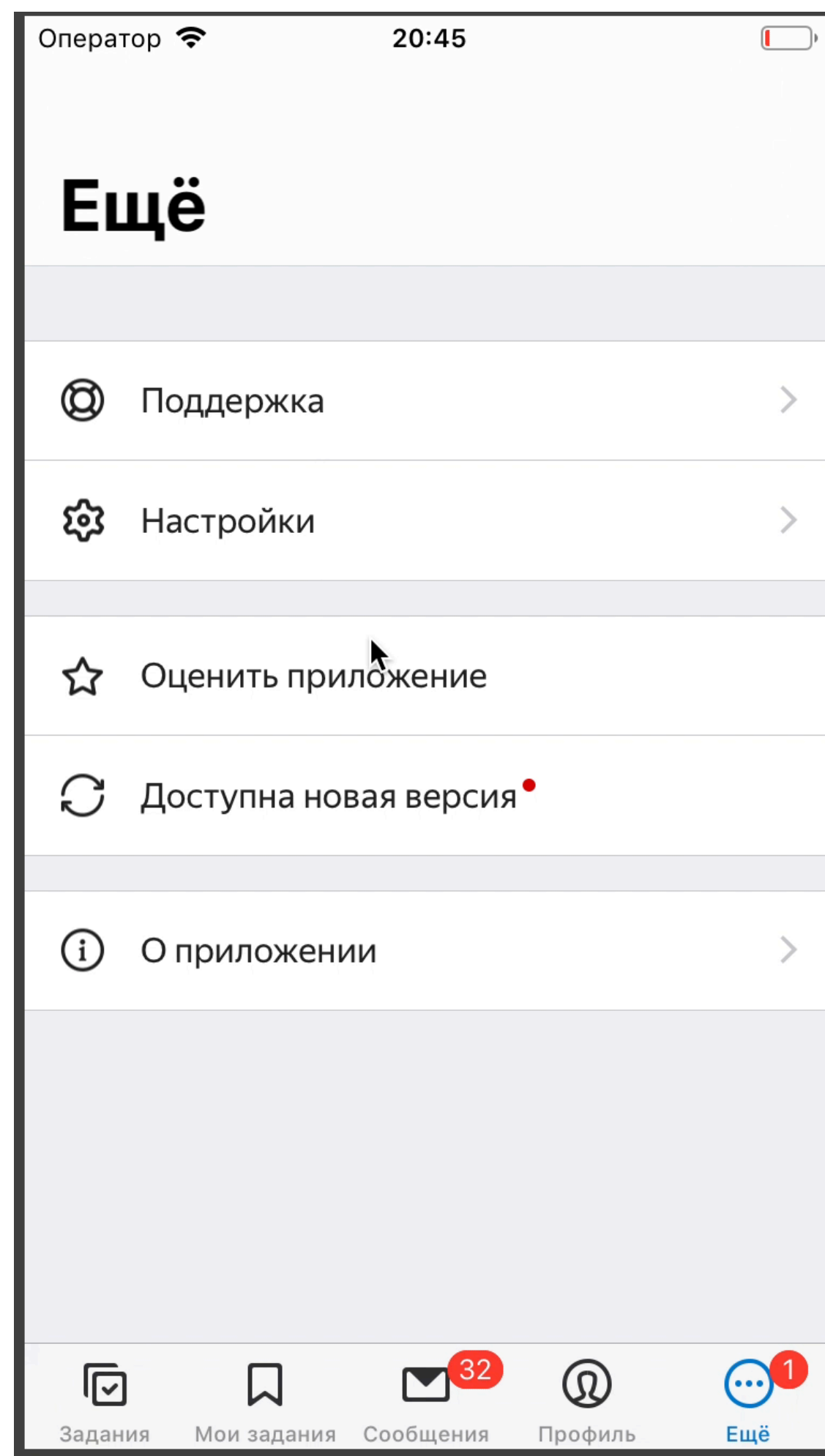
Невосстановимые ошибки

Варианты отображения в зависимости от сценария использования



Нерегистрируемые ошибки и проблемы

Больше каналов обратной связи — меньше жалоб в App Store



- work_lifecycle
- task
- task_done
- feedback_disapointment_low_prices
- feedback_disapointment_no_tasks
- feedback_disapointment_withdraw_issues
- feedback_disapointment_rejected_tasks
- feedback_disapointment_slow_approvement
- feedback_disapointment_slow_loading
- feedback_disapointment_difficult_instructions
- feedback_disapointment_inconvenient_interface

Итого: забота о “пострадавших”

- › Информативность – сообщаем на понятном для пользователя языке
- › Полезность – удерживаем пользователя в его сценарии работы
- › Надежда – способ конструктивно пожаловаться
- › Опрос – классификация проблем, не определяемых из кода

Учимся на ошибках



Анализ ошибок: Yandex AppMetrica



Applications Trackers Media Sources Push Campaigns Blog Documentation

All users + Add segment

Conditions Traffic sources Push campaign Users Devices Application **Network** segmented 25.50% of all users

Carrier

Connection type Cellular

Network type

Search

Selection Include Exclude

Wi-Fi

Cellular

Name the segment, visible for you only

iOS and Cellular



Refresh Report Export

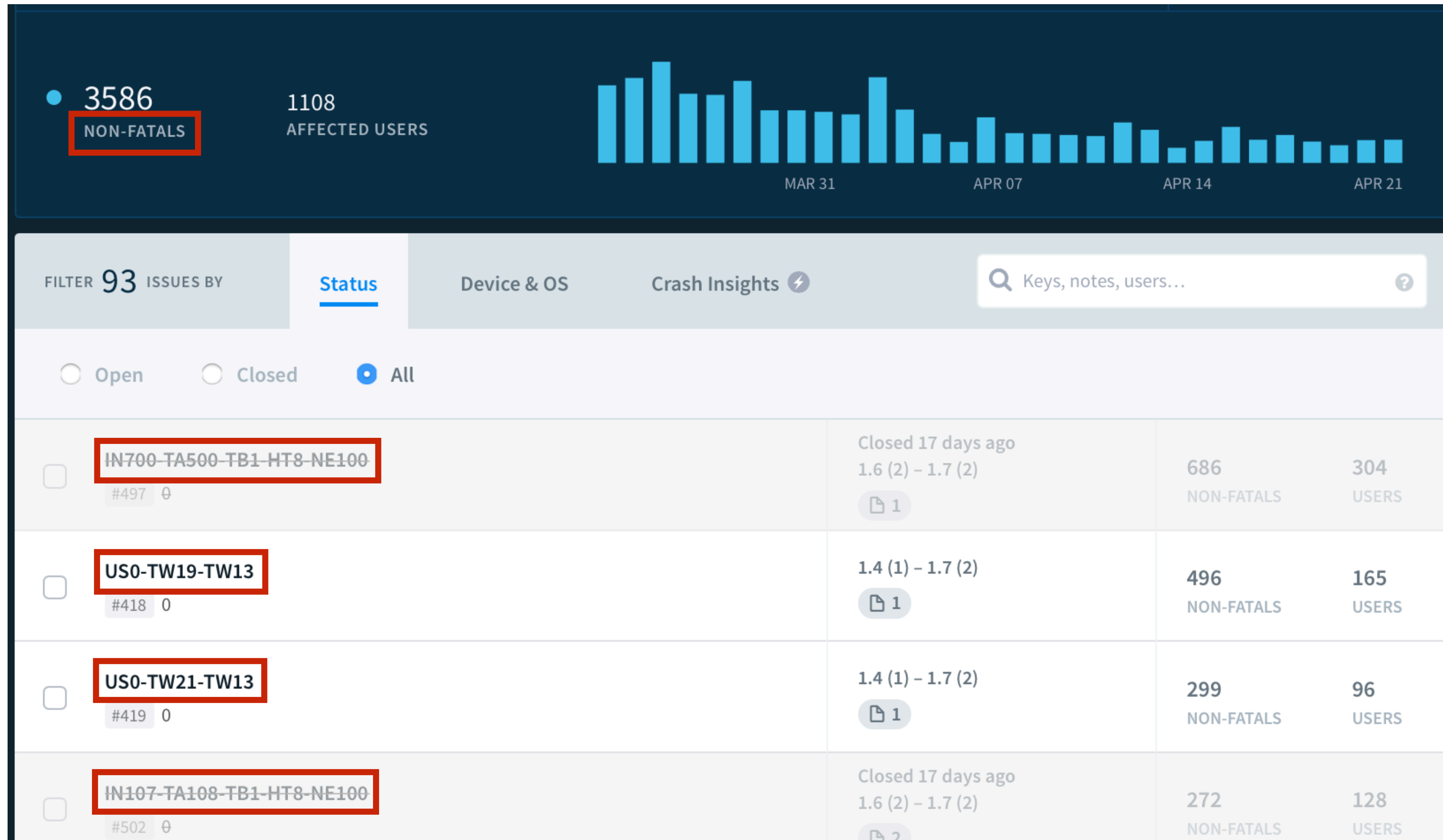


Анализ ошибок: Yandex AppMetrica



- › Хранение точных данных за долгий период
- › Анализ ошибок в разрезах моделей устройств, вида сетевого соединения, версии iOS, версий приложения и т.д.
- › Быстрые и точные графики и диаграммы
- › Выгрузка сырых данных за любой период для ручного разбора

Мониторинг ошибок: Non-Fatals в Crashlytics



Группировка по своему ключу в Crashlytics

- › iOS (подменяем домен NSError)

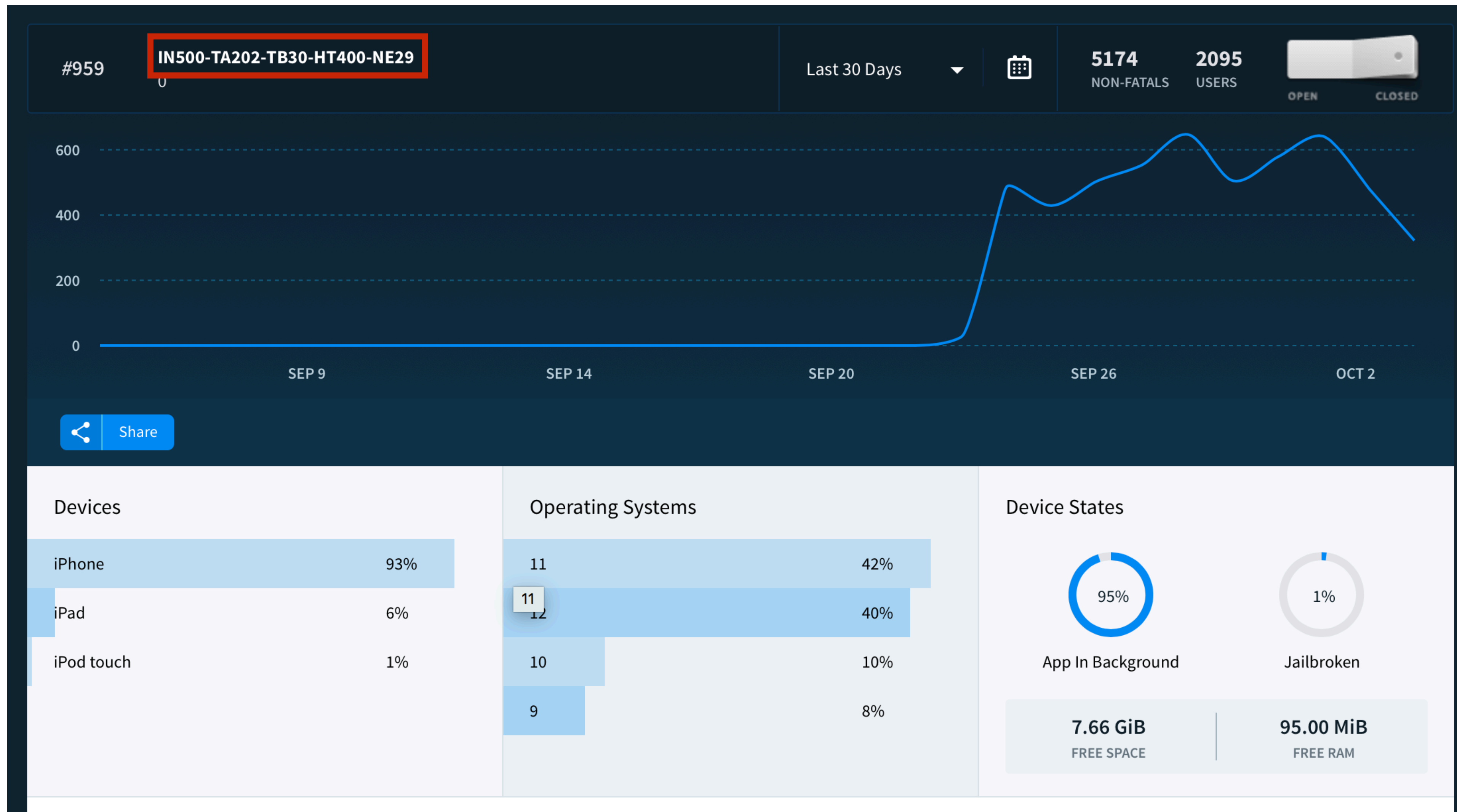
```
let codesChain = error.shortCodesChain
let crashlyticsError = NSError(domain: codesChain code: 0, userInfo: params)
crashlytics.recordError(nonFatalCrashlyticsError)
```

- › Android (подменяем стек-трейс у самого вложенного эксепшена)

```
Throwable lastCause = ex;
while (lastCause.getCause() != null) {
    lastCause = lastCause.getCause();
}
final StackTraceElement[] stackTrace = lastCause.getStackTrace();
final StackTraceElement[] newStackTrace = new
StackTraceElement[stackTrace.length + 1];
System.arraycopy(stackTrace, 0, newStackTrace, 1, stackTrace.length);

final String traceCode = ex.traceCode();
final String extTraceCode = ex.extTraceCode();
newStackTrace[0] = new StackTraceElement(extTraceCode, "", traceCode, 0);
lastCause.setStackTrace(newStackTrace);
```






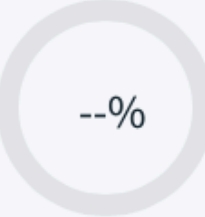
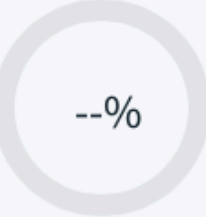

Разбор Non-Fatal в Crashlytics



Разбор Non-Fatal в Crashlytics

#959 **IN500-TA202-TB30-HT400-NE29**

◀ Oct 04 2018 22:33 (UTC) ▶
Version 1.12 (1) All Versions ▾

Device  iPhone 5s DEVICE  Portrait ORIENTATION	Operating System  iOS 11.4.1 (15G77) OS VERSION  Portrait UI ORIENTATION  No JAILBROKEN	Device Statistics  --% Ram Free  --% Disk Free
---	--	---

Keys Logs

Search Keys...

Key	Value
error_codes_stack	InteractorError.getMapPins-TolokaAPIError.taskSuites-TolokaBackendError.validationFailure(ValidationError.valueOutOfBounds)-HTTPError.no400BadRequest-NetworkError.responseValidationFailed

Нотификации Crashlytics

Stability Digest for Apr 2, 2018

Yang iOS
ru.yandex.mobile.yang.inhouse.prd

TRENDING TOP ISSUES

#63
US0-TW20-TW13
0

1.5 22 non-fatals 1 user

[View Dashboard](#)

Toloka iOS 1.7 (2)
ru.yandex.mobile.toloka

NEW ISSUE

Heads up! We detected a new non-fatal issue in IN107-TA108-TB1-HT6-NE29

[Learn more](#)

#607
IN107-TA108-TB1-HT6-NE29
0

Toloka iOS 1.7 (2)
ru.yandex.mobile.toloka

REGRESSED ISSUE

A non-fatal issue was closed, but it popped up again in version 1.7 (2).

[Find out why](#)

#525
US8-IN312-TA302-TB1-HT8-NE100
0

Мониторинг ошибок: Non-Fatals в Crashlytics

- › Наглядный real-time мониторинг
- › Уведомления о новых видах ошибок, о резко растущих видах ошибок
- › Возможность пометить ошибки как решенные
- › Прикрепляется stack-trace и описание девайса
- › Ограничение: 8 ошибок за сессию

Итого: мониторинг и анализ

- › Полнота – использование нескольких систем трекинга
- › Гибкость – возможность использовать свою систему идентификации
- › Скорость – время доставки данных, умные нотификации

Заключение




Общая стратегия разбора ошибок

- › Работа с идентификаторами ошибочных сценариев
- › Разбор частотных (топ-N по количеству)
- › Разбор критичных (топ-N по жалобам)
- › Долгосрочное отслеживание идентификаторов
- › Постоянство разбора ошибок

Показатели хорошей обработки ошибок

- › Легко для разработчика: единообразно и безопасно
- › Безболезненно для пользователя: понятные и восстанавливаемые
- › Эффективно для команды: удобный мониторинг и анализ
- › Полезно для проекта: быстрые циклы багфиксов



**Никогда нельзя недоценивать
предсказуемость тупизны.**

из к/ф Большой Куш

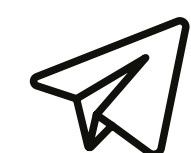
Спасибо!

Дмитрий Михайлов

Руководитель мобильной разработки Яндекс.Толоки



dmitry-mik@yandex-team.ru



@dima_clif