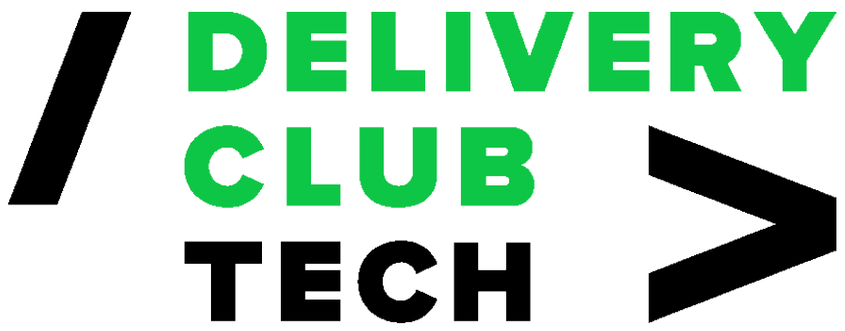
DELIVERY
CLUB
TECH

Как переписать сетевой слой так, чтобы его не пришлось переписывать снова

Аносов Александр



Как переписать сетевой слой так, чтобы его не пришлось переписывать снова

Аносов Александр

← это я

DELIVERY CLUB TECH

Предисловие

Я и сетевой слой

это я →

↖ это сетевой слой

netPrint.ru

- 2015-2017
- Приложение с нуля
- Objective-C
- Один бэкенд
- Менеджер со статическими методами
- NSURLConnection, потом NSURLSession

netPrint.ru

- 2015-2017
- Приложение с нуля
- Objective-C
- Один бэкенд
- Менеджер со статическими методами
- NSURLConnection, потом NSURLSession

netPrint.ru

- 2015-2017
- Приложение с нуля
- Objective-C
- Один бэкенд
- Менеджер со статическими методами
- NSURLConnection, потом NSURLSession

netPrint.ru

- 2015-2017
- Приложение с нуля
- Objective-C
- Один бэкенд
- Менеджер со статическими методами
- NSURLConnection, потом NSURLSession

netPrint.ru

- 2015-2017
- Приложение с нуля
- Objective-C
- Один бэкенд
- Менеджер со статическими методами
- NSURLConnection, потом NSURLSession

netPrint.ru

- 2015-2017
- Приложение с нуля
- Objective-C
- Один бэкенд
- Менеджер со статическими методами
- NSURLConnection, потом NSURLSession

Простите

ДомКлик

- 2018-2019
- Приложению 2 года
- Swift
- Переход от мобильного бэка к микросервисам
- Переписали сетевой слой из-за сложности
- URLSession в старом и в новом

ДомКлик

- 2018-2019
- Приложению 2 года
- Swift
- Переход от мобильного бэка к микросервисам
- Переписали сетевой слой из-за сложности
- URLSession в старом и в новом

ДомКлик

- 2018-2019
- Приложению 2 года
- Swift
- Переход от мобильного бэка к микросервисам
- Переписали сетевой слой из-за сложности
- URLSession в старом и в новом

ДомКлик

- 2018-2019
- Приложению 2 года
- Swift
- Переход от мобильного бэка к микросервисам
- Переписали сетевой слой из-за сложности
- URLSession в старом и в новом

ДомКлик

- 2018-2019
- Приложению 2 года
- Swift
- Переход от мобильного бэка к микросервисам
- Переписали сетевой слой из-за сложности
- URLSession в старом и в новом

ДомКлик

- 2018-2019
- Приложению 2 года
- Swift
- Переход от мобильного бэка к микросервисам
- Переписали сетевой слой из-за сложности
- URLSession в старом и в новом

Степан, Гена, привет

МойСклад

- 2019-2020
- Приложению 3 года
- Swift
- Добавление новой версии API
- Менеджер со статическими методами, переписали
- Alamofire в старом и в новом

МойСклад

- 2019-2020
- Приложению 3 года
- Swift
- Добавление новой версии API
- Менеджер со статическими методами, переписали
- Alamofire в старом и в новом

МойСклад

- 2019-2020
- Приложению 3 года
- Swift
- Добавление новой версии API
- Менеджер со статическими методами, переписали
- Alamofire в старом и в новом

МойСклад

- 2019-2020
- Приложению 3 года
- Swift
- Добавление новой версии API
- Менеджер со статическими методами, переписали
- Alamofire в старом и в новом

МойСклад

- 2019-2020
- Приложению 3 года
- Swift
- Добавление новой версии API
- Менеджер со статическими методами, переписали
- Alamofire в старом и в новом

МойСклад

- 2019-2020
- Приложению 3 года
- Swift
- Добавление новой версии API
- Менеджер со статическими методами, переписали
- Alamofire в старом и в новом

Delivery Club

- 2020-…
- Приложению 7 лет
- Objective-C + Swift
- Три бэкенда за которыми микросервисы
- Переписали сетевой слой, расскажу почему
- AFNetworking у старого, URLSession у нового

Delivery Club

- 2020-…
- Приложению 7 лет
- Objective-C + Swift
- Три бэкенда за которыми микросервисы
- Переписали сетевой слой, расскажу почему
- AFNetworking у старого, URLSession у нового

Delivery Club

- 2020-…
- Приложению 7 лет
- Objective-C + Swift
- Три бэкенда за которыми микросервисы
- Переписали сетевой слой, расскажу почему
- AFNetworking у старого, URLSession у нового

Delivery Club

- 2020-…
- Приложению 7 лет
- Objective-C + Swift
- Три бэкенда за которыми микросервисы
- Переписали сетевой слой, расскажу почему
- AFNetworking у старого, URLSession у нового

Delivery Club

- 2020-…
- Приложению 7 лет
- Objective-C + Swift
- Три бэкенда за которыми микросервисы
- Переписали сетевой слой, расскажу почему
- AFNetworking у старого, URLSession у нового

Delivery Club

- 2020-…
- Приложению 7 лет
- Objective-C + Swift
- Три бэкенда за которыми микросервисы
- Переписали сетевой слой, расскажу почему
- AFNetworking у старого, URLSession у нового

Саша, Паша, привет

О чем я расскажу

- Интересная история
- Причины переписывания сетевого слоя
- Хорошие практики

О чем я расскажу

- Интересная история
- Причины переписывания сетевого слоя
- Хорошие практики

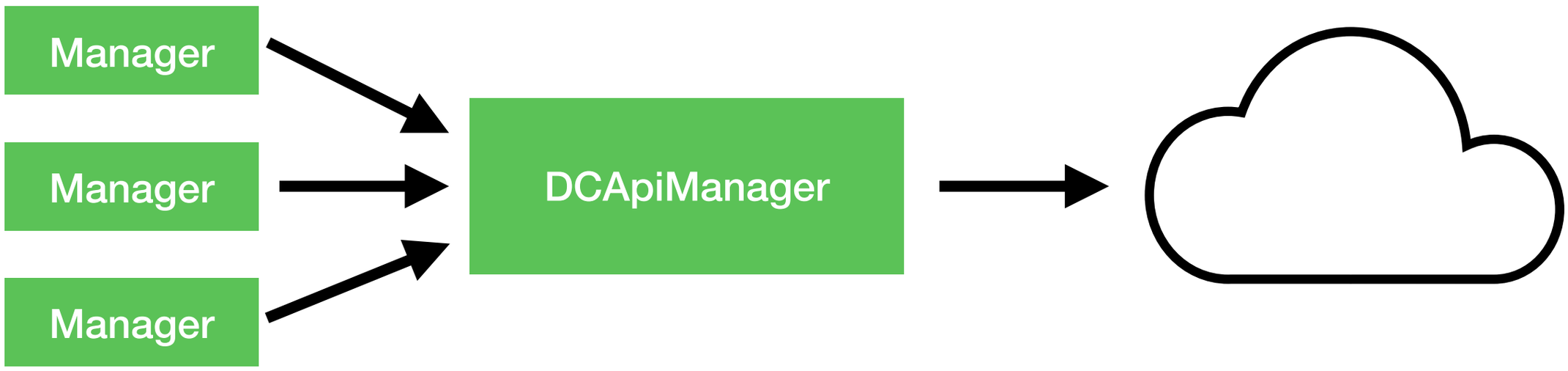
О чем я расскажу

- Интересная история
- Причины переписывания сетевого слоя
- Хорошие практики

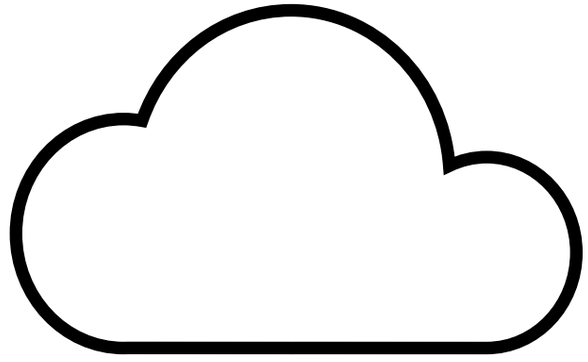
Как все начиналось


Баг с
разлогином!

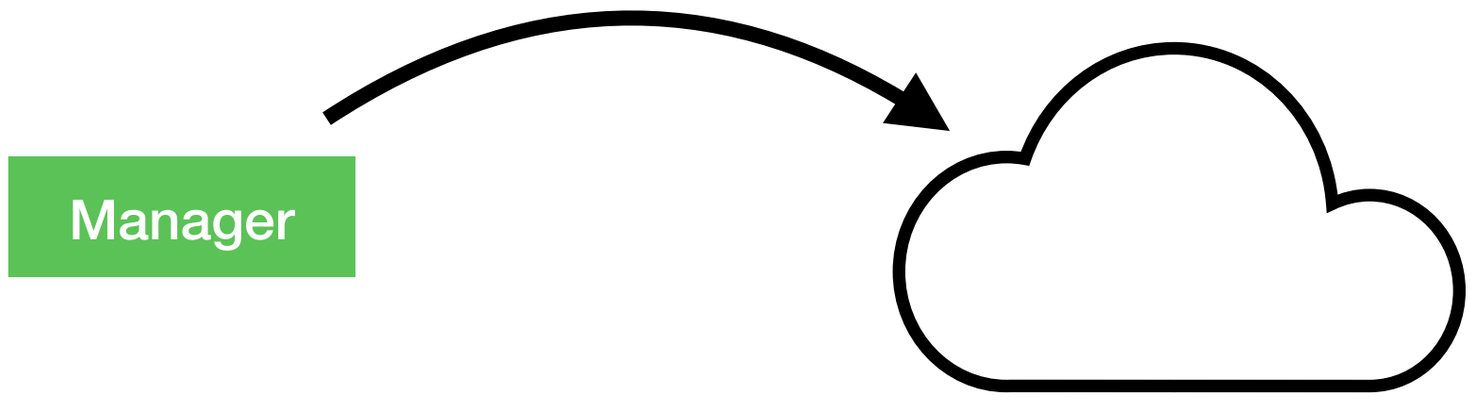
Баг с разлогином



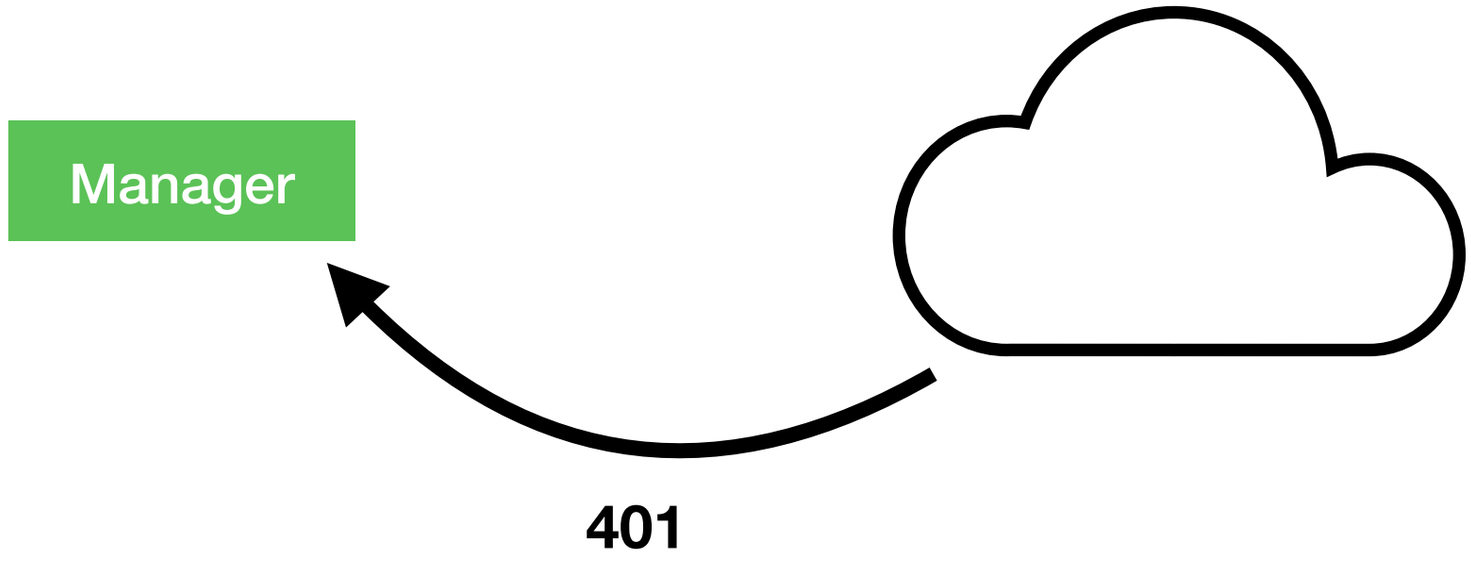
Баг с разлогином



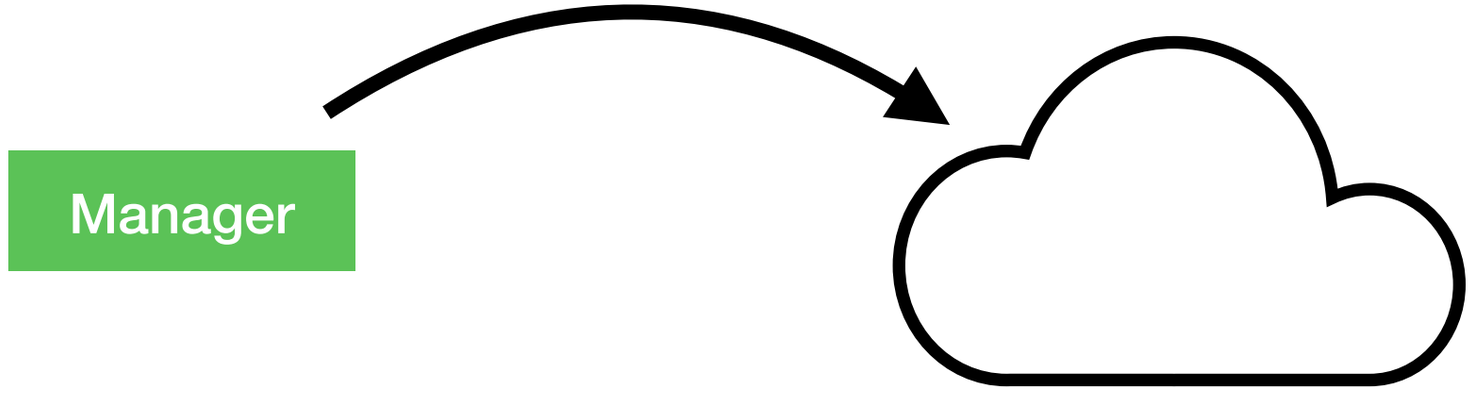
Баг с разлогином



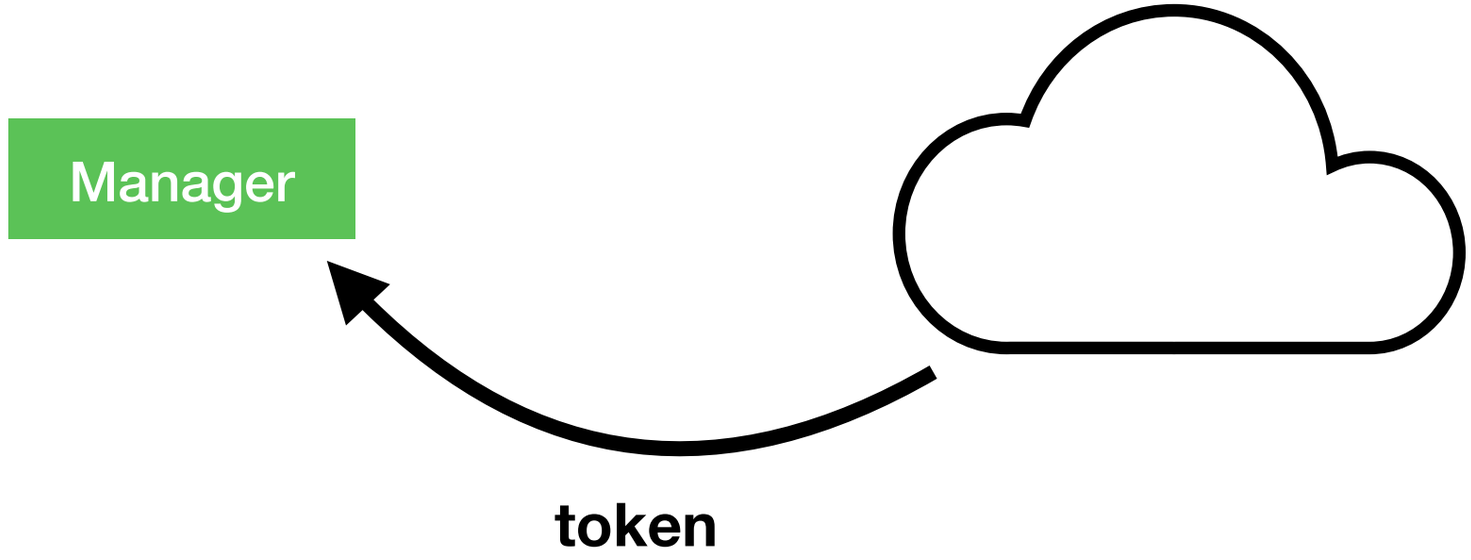
Баг с разлогином



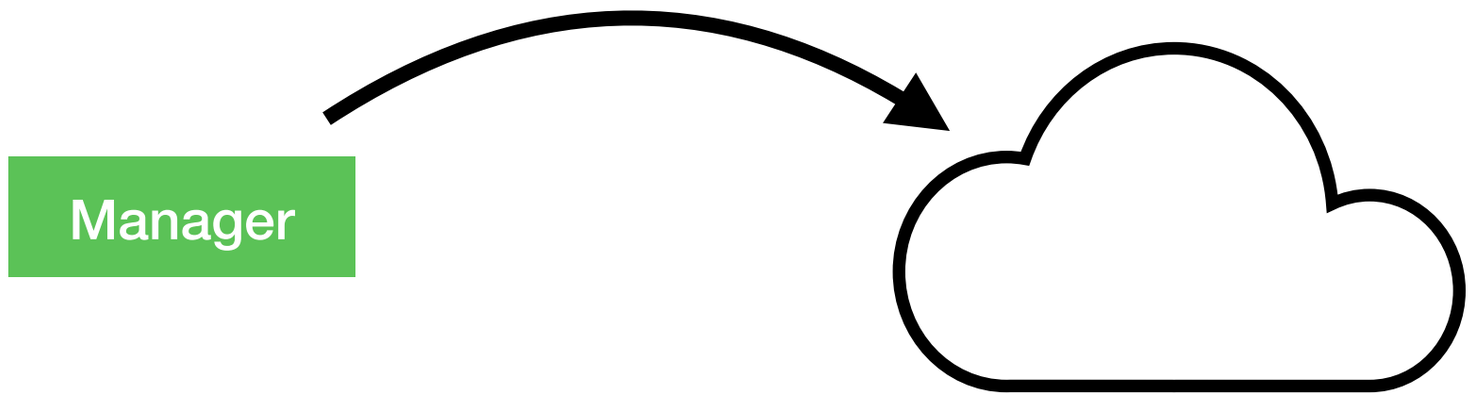
Баг с разлогином



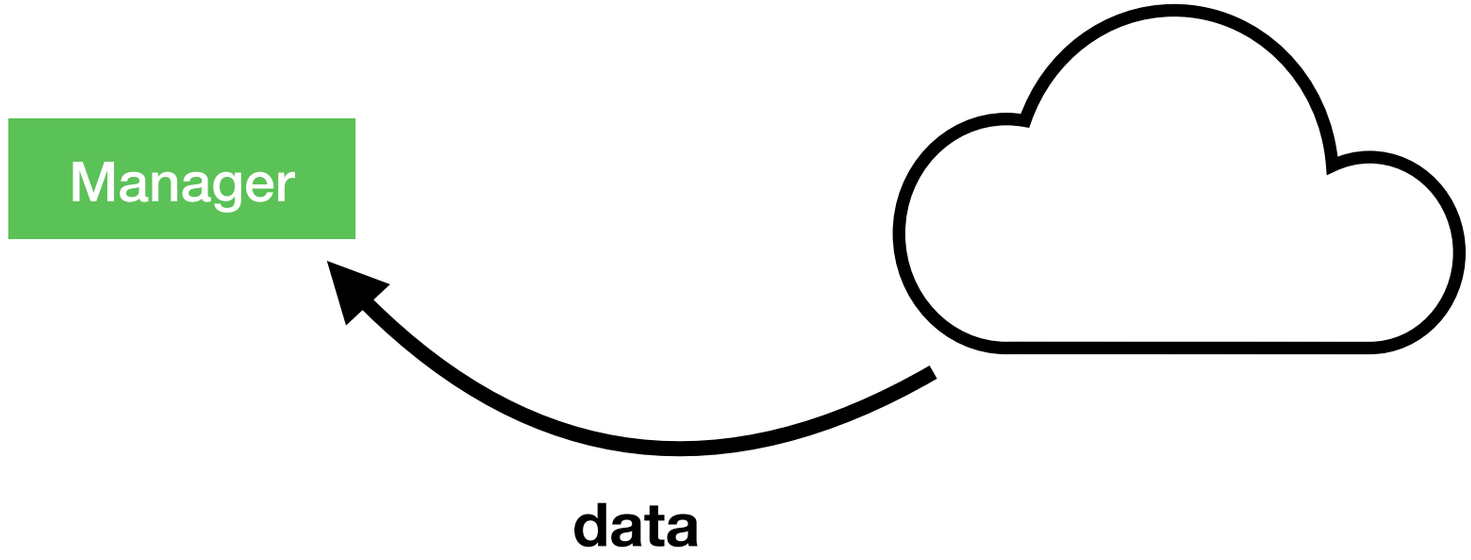
Баг с разлогином



Баг с разлогином

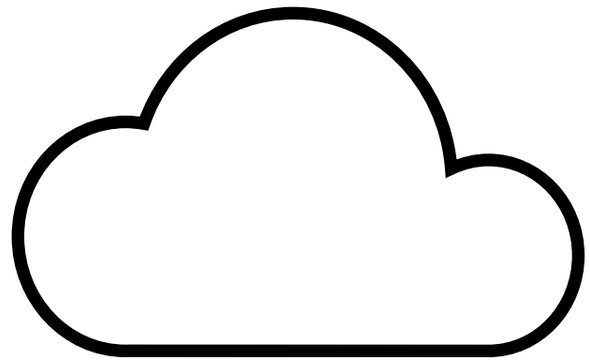


Баг с разлогином



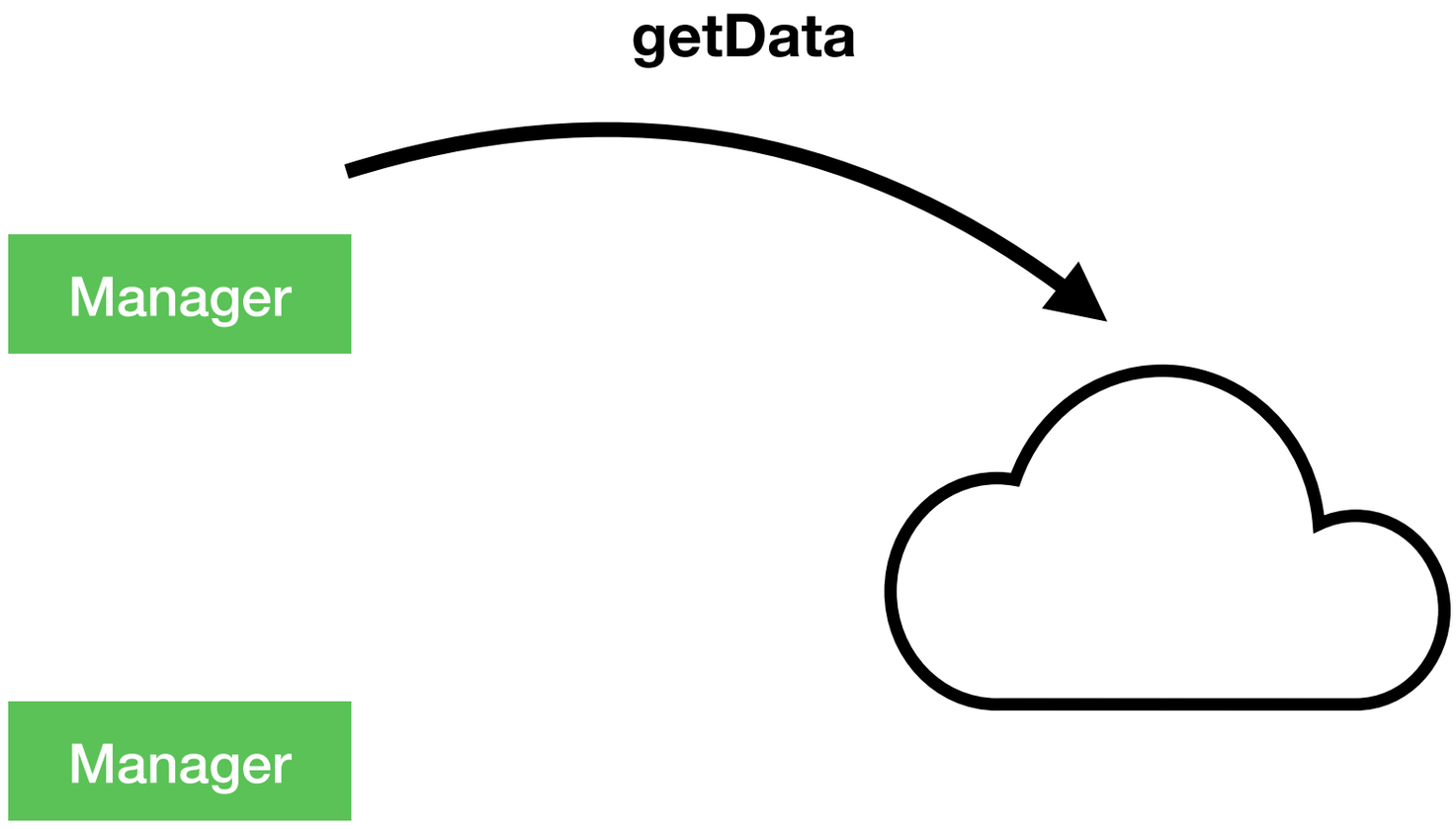
Баг с разлогином

Manager

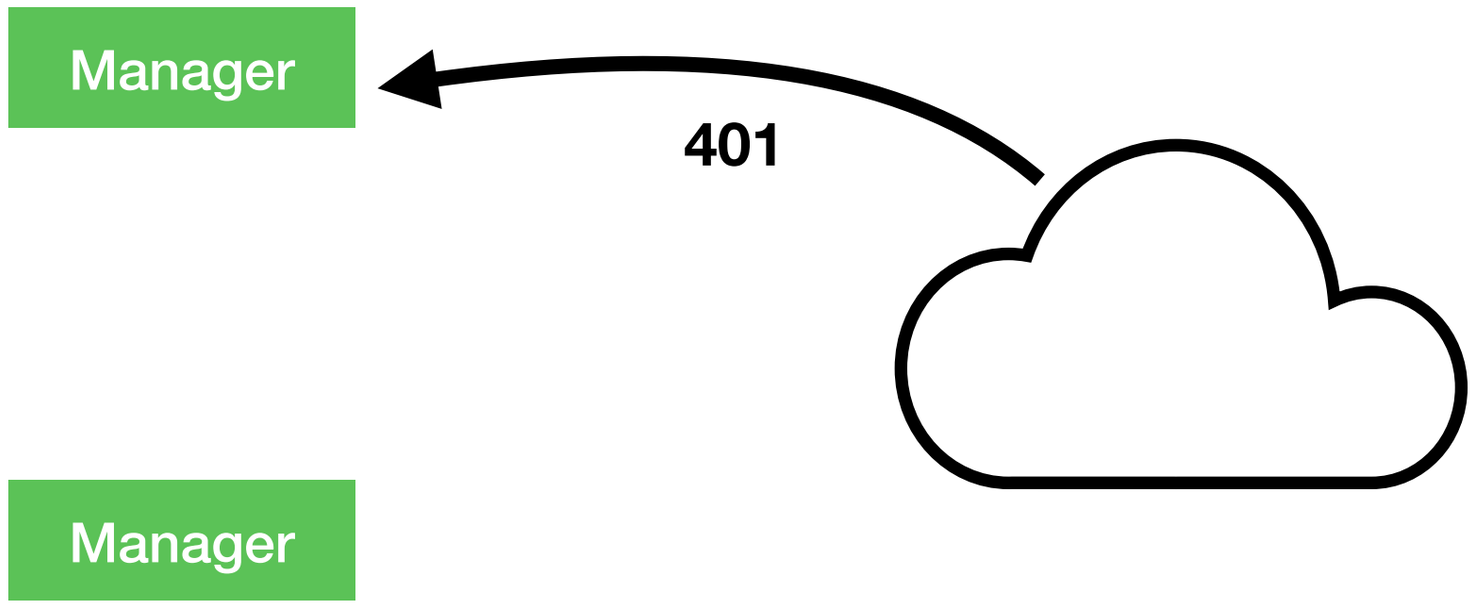


Manager

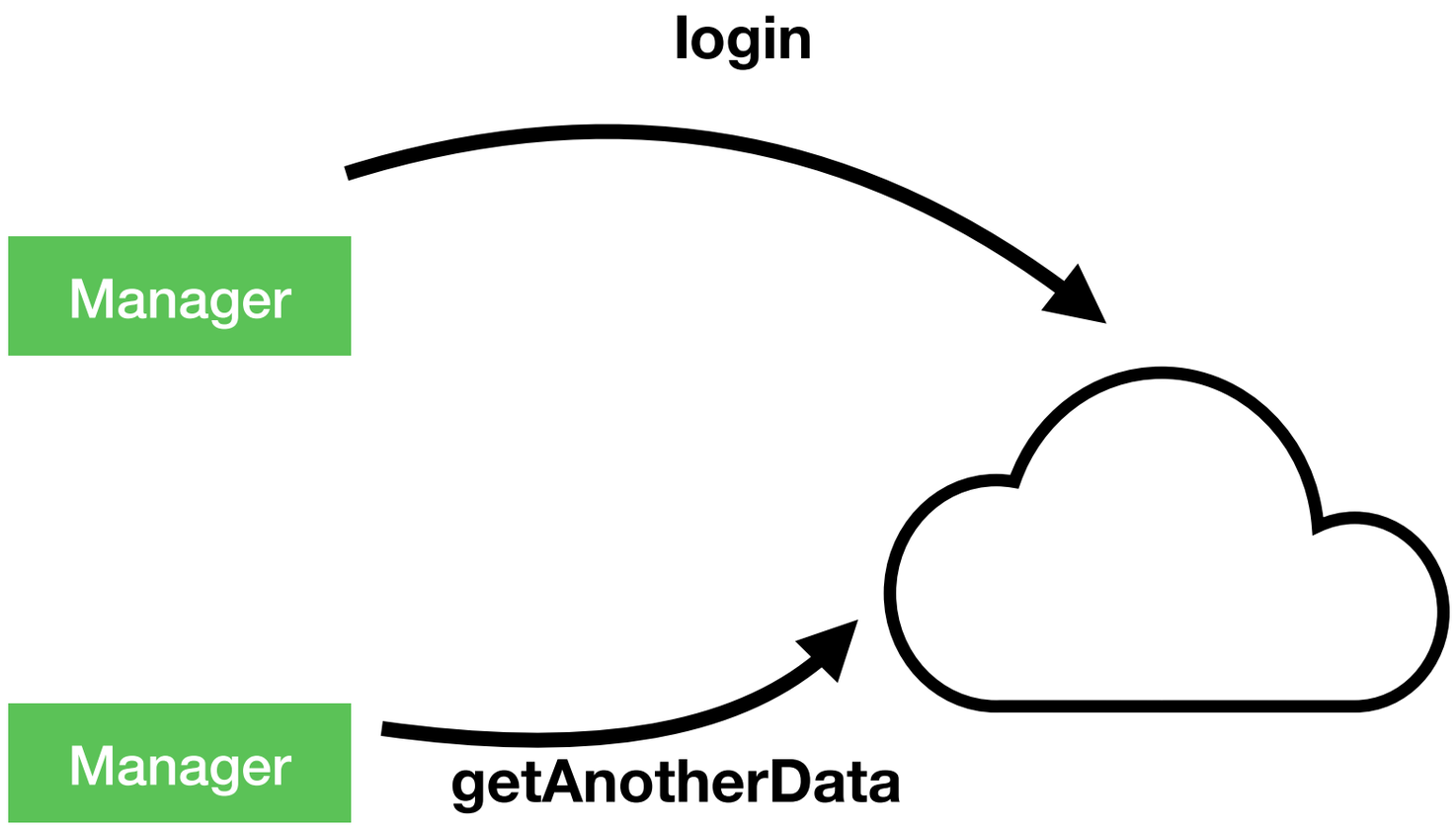
Баг с разлогином



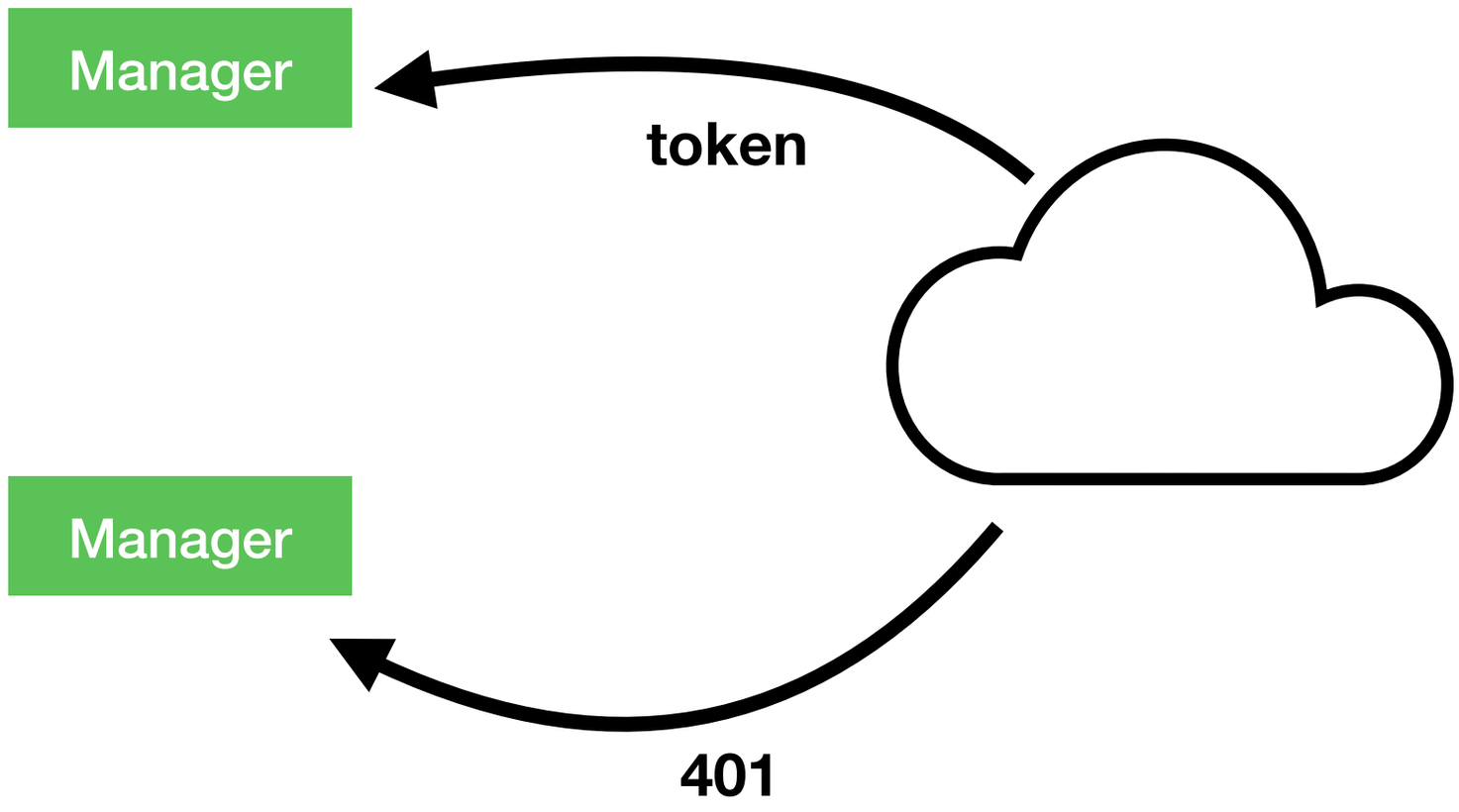
Баг с разлогином



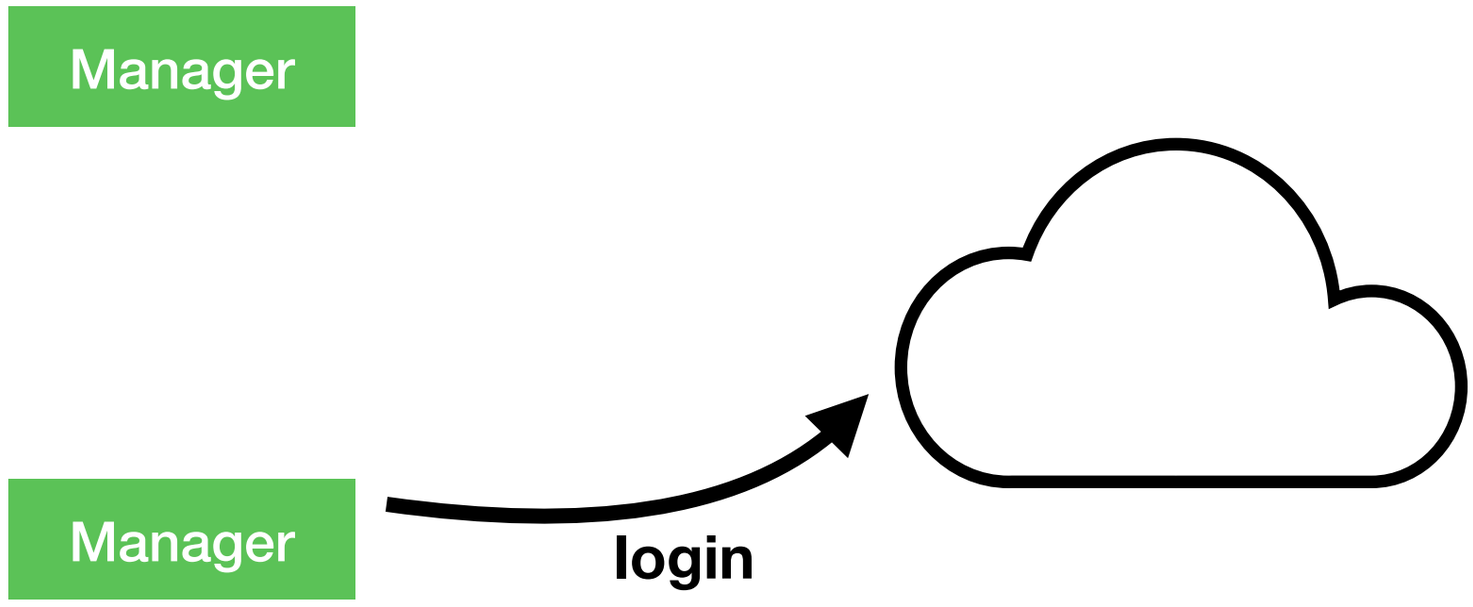
Баг с разлогином



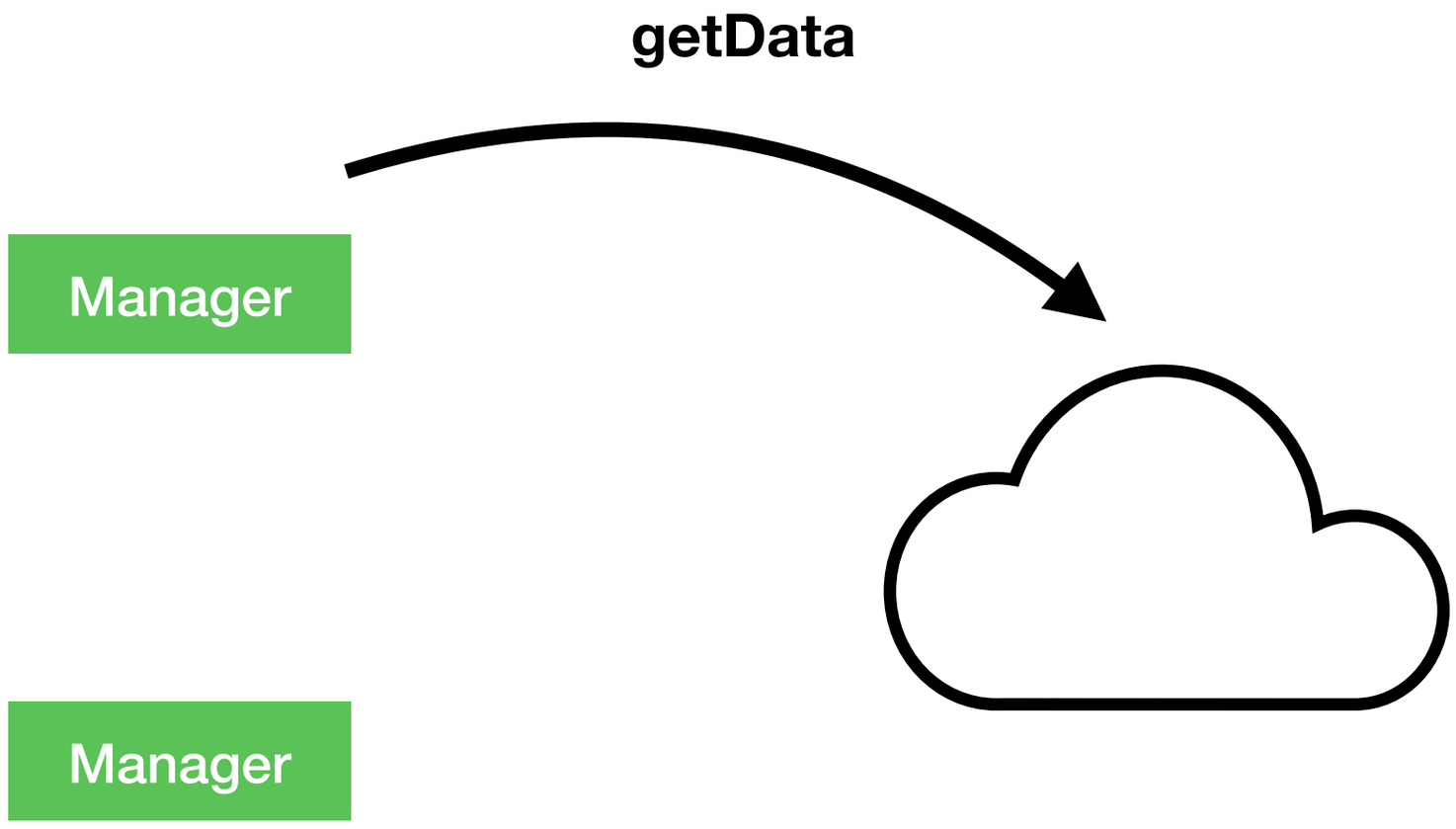
Баг с разлогином



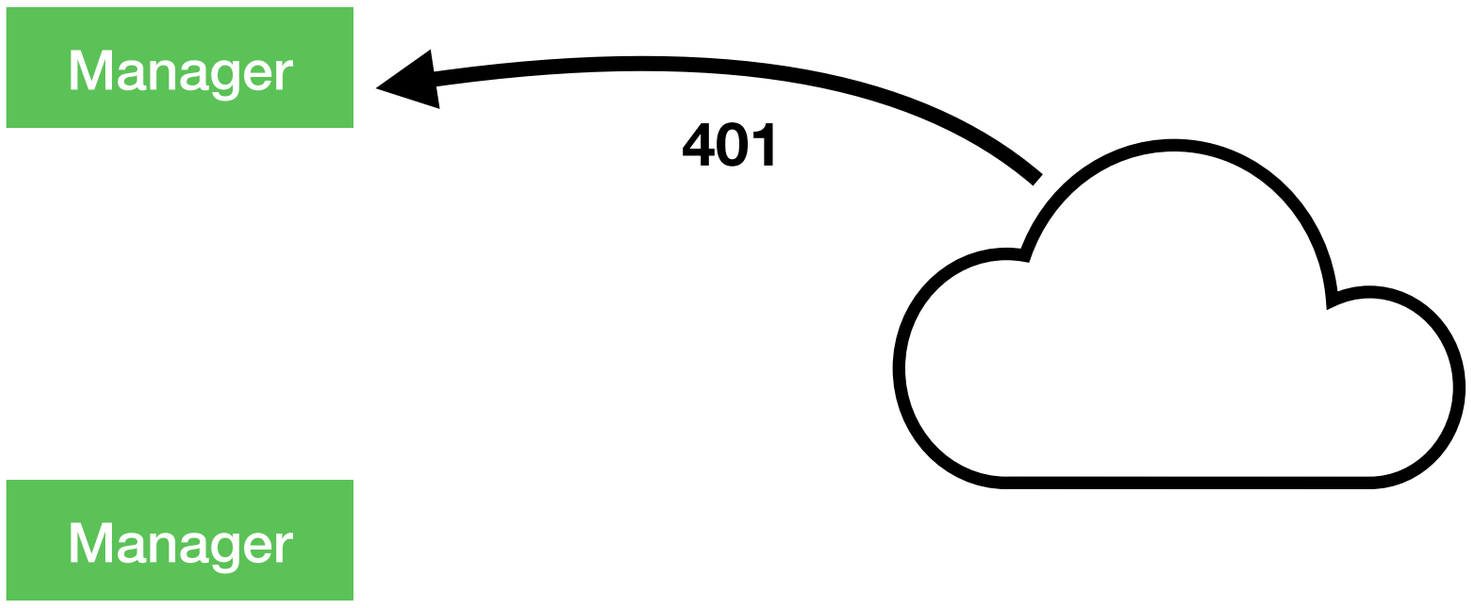
Баг с разлогином



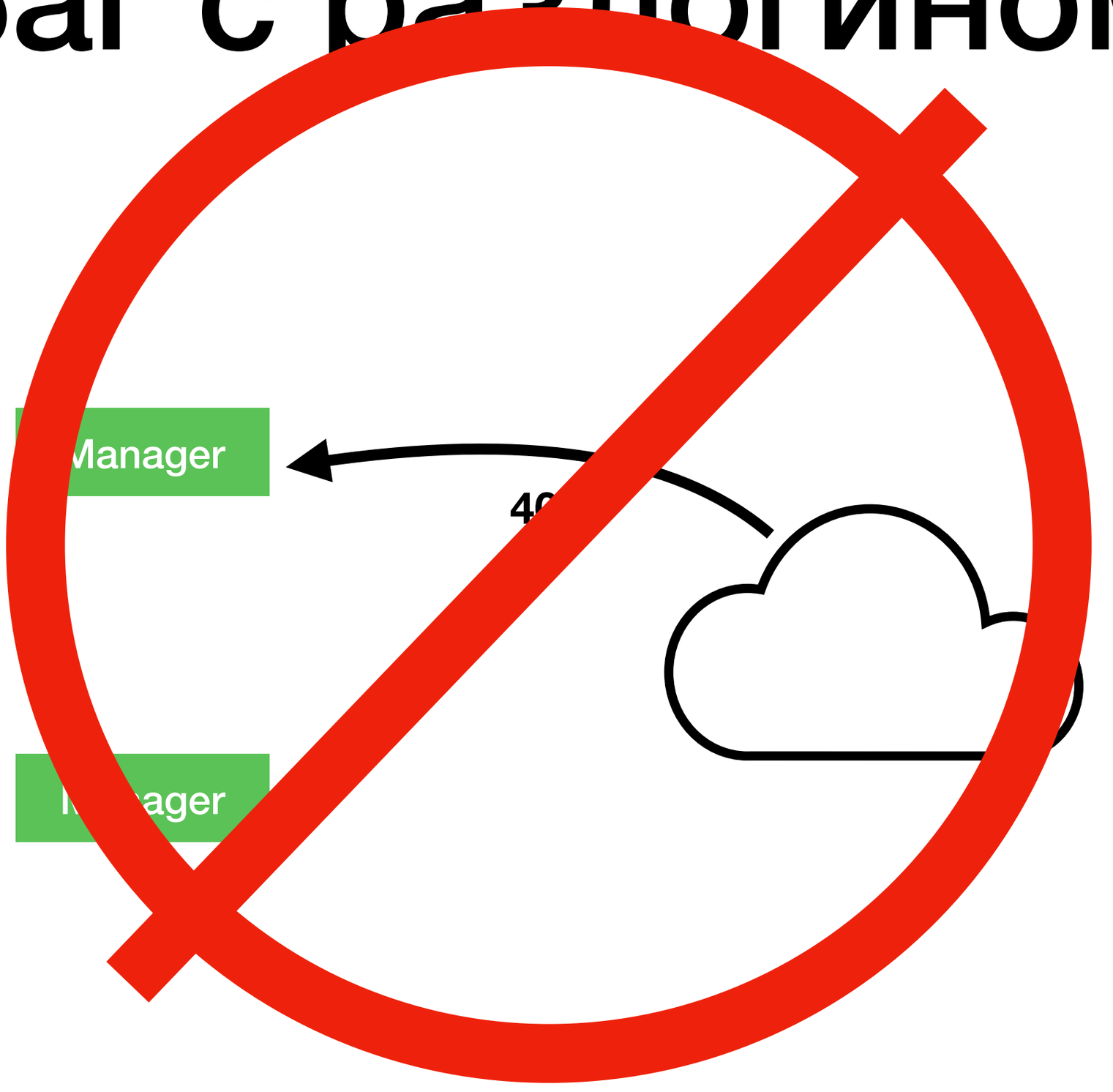
Баг с разлогином



Баг с разлогином



Баг с разлогином



Решение

Нужно добавить очередь запросов…

Решение

Нужно добавить очередь запросов…

А заодно все переписать

Старый сетевой слой

- 16 Objective-C файлов + 2 Swift extension
- DCApiManager на 600 строк
- DCApiManagerHelper на 300 строк
- DCErrorHandler на 300 строк
- AFNetworking
- ~170 методов Api

Старый сетевой слой

- 16 Objective-C файлов + 2 Swift extension
- DCApiManager на 600 строк
- DCApiManagerHelper на 300 строк
- DCErrorHandler на 300 строк
- AFNetworking
- ~170 методов Api

Старый сетевой слой

- 16 Objective-C файлов + 2 Swift extension
- DCApiManager на 600 строк
- DCApiManagerHelper на 300 строк
- DCErrorHandler на 300 строк
- AFNetworking
- ~170 методов Api

Старый сетевой слой

- 16 Objective-C файлов + 2 Swift extension
- DCApiManager на 600 строк
- DCApiManagerHelper на 300 строк
- DCErrorHandler на 300 строк
- AFNetworking
- ~170 методов Api

Старый сетевой слой

- 16 Objective-C файлов + 2 Swift extension
- DCApiManager на 600 строк
- DCApiManagerHelper на 300 строк
- DCErrorHandler на 300 строк
- AFNetworking
- ~170 методов Api

Старый сетевой слой

- 16 Objective-C файлов + 2 Swift extension
- DCApiManager на 600 строк
- DCApiManagerHelper на 300 строк
- DCErrorHandler на 300 строк
- AFNetworking
- ~170 методов Api

План

- Закрываем протоколом
- Пишем рядом новую реализацию
- Закрываем фиче-тогглом
- Постепенно раскатываем

План

- Закрываем протоколом
- Пишем рядом новую реализацию
- Закрываем фиче-тогглом
- Постепенно раскатываем

План

- Закрываем протоколом
- Пишем рядом новую реализацию
- Закрываем фиче-тогглом
- Постепенно раскатываем

План

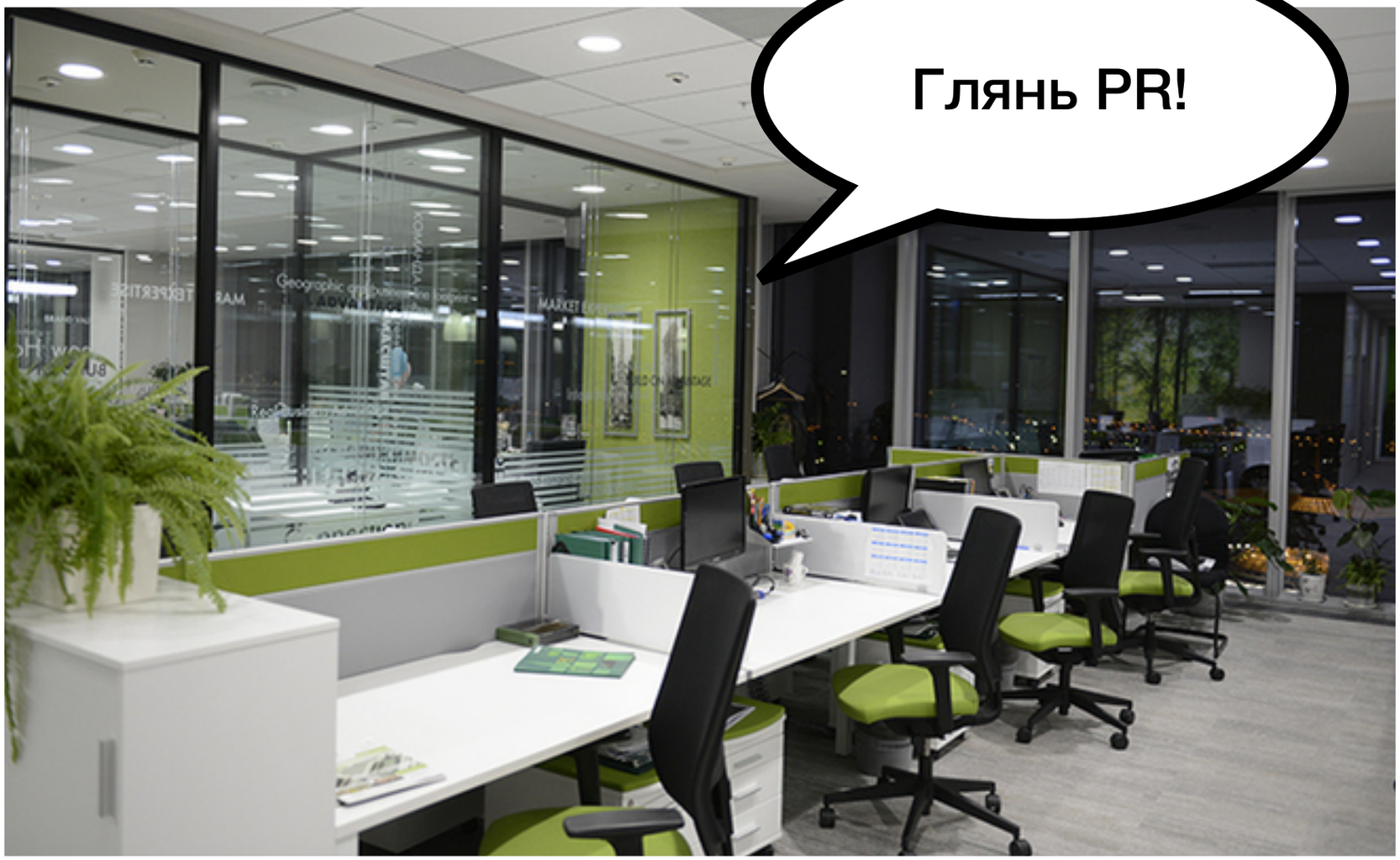
- Закрываем протоколом
- Пишем рядом новую реализацию
- Закрываем фиче-тогглом
- Постепенно раскатываем

А реализовать этот план взялся Дима

О чем я расскажу

- Интересная история
- ~~Причины переписывания сетевого слоя~~ 
- Хорошие практики

Что было дальше


Глянь PR!

Первый PR*

Первый PR*

- HTTPClient vs HttpClient

Первый PR*

- HTTPClient vs HttpClient

Первый PR*

- HTTPClient vs HttpClient
- APIManager vs ApiManager

Первый PR*

- HTTPClient vs HttpClient
- APIManager vs ApiManager
- HttpClient vs ApiManager

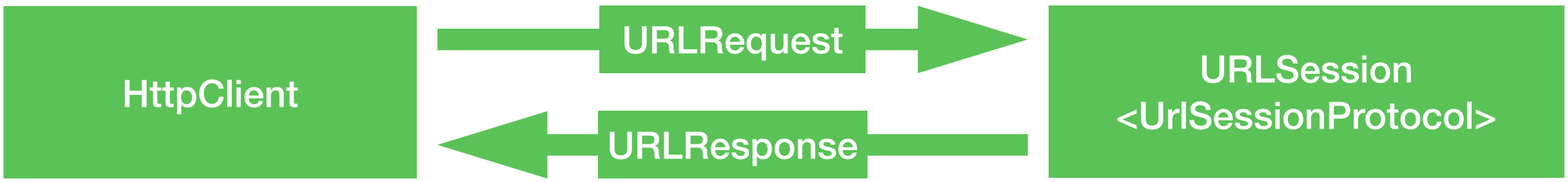
Кусочек сетевого слоя

URLSession

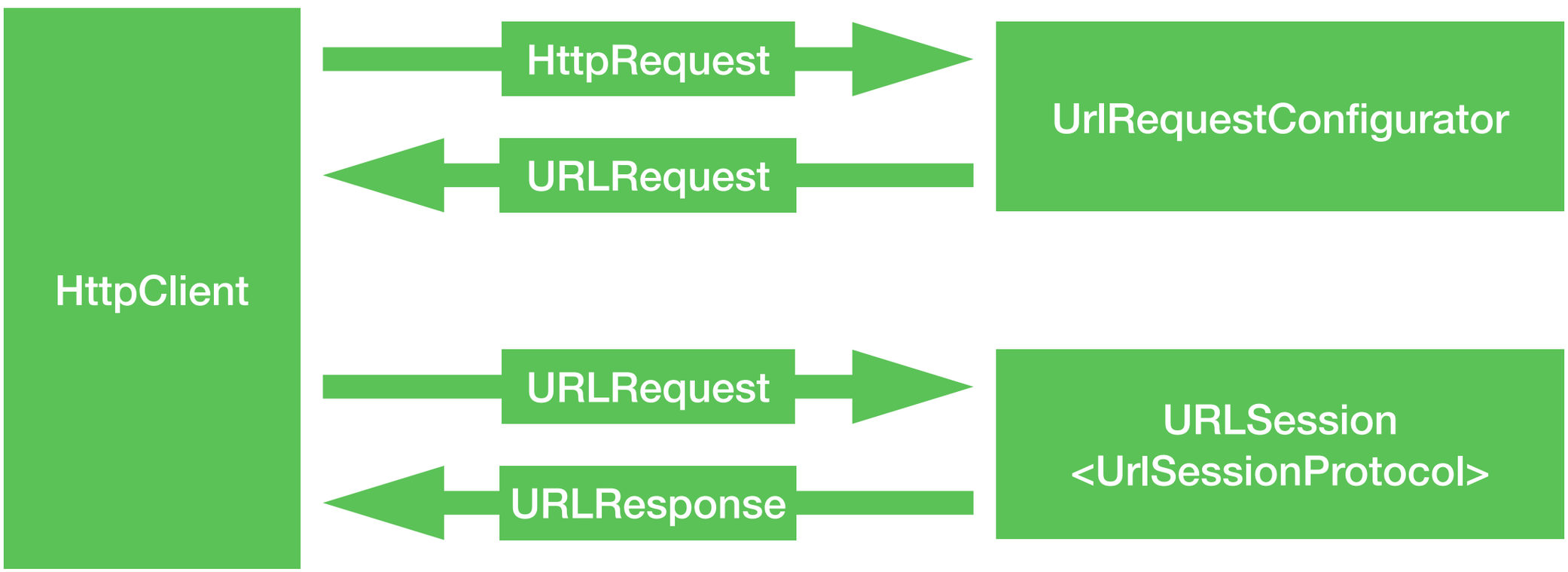
Кусочек сетевого слоя

URLSession
<UrlSessionProtocol>

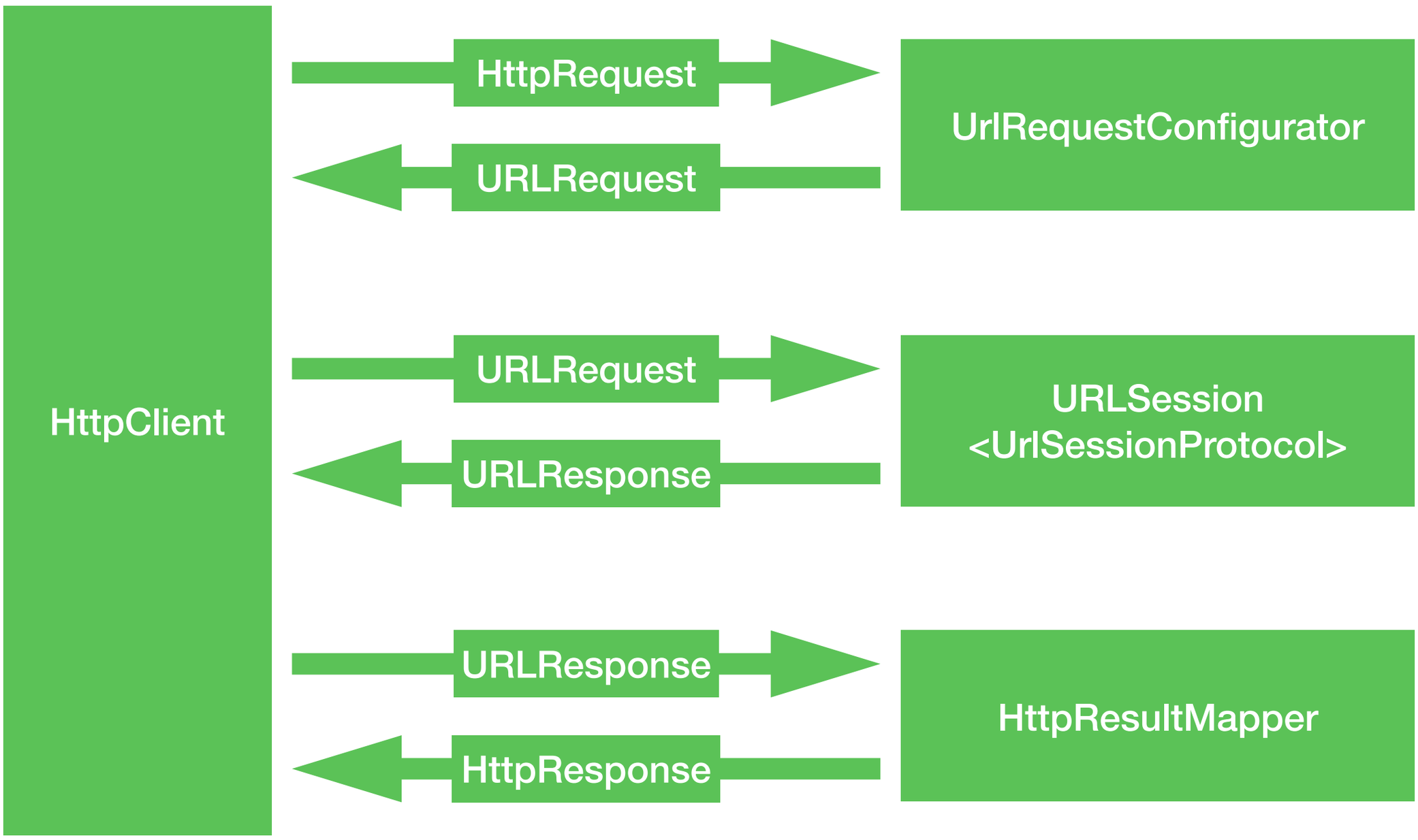
Кусочек сетевого слоя



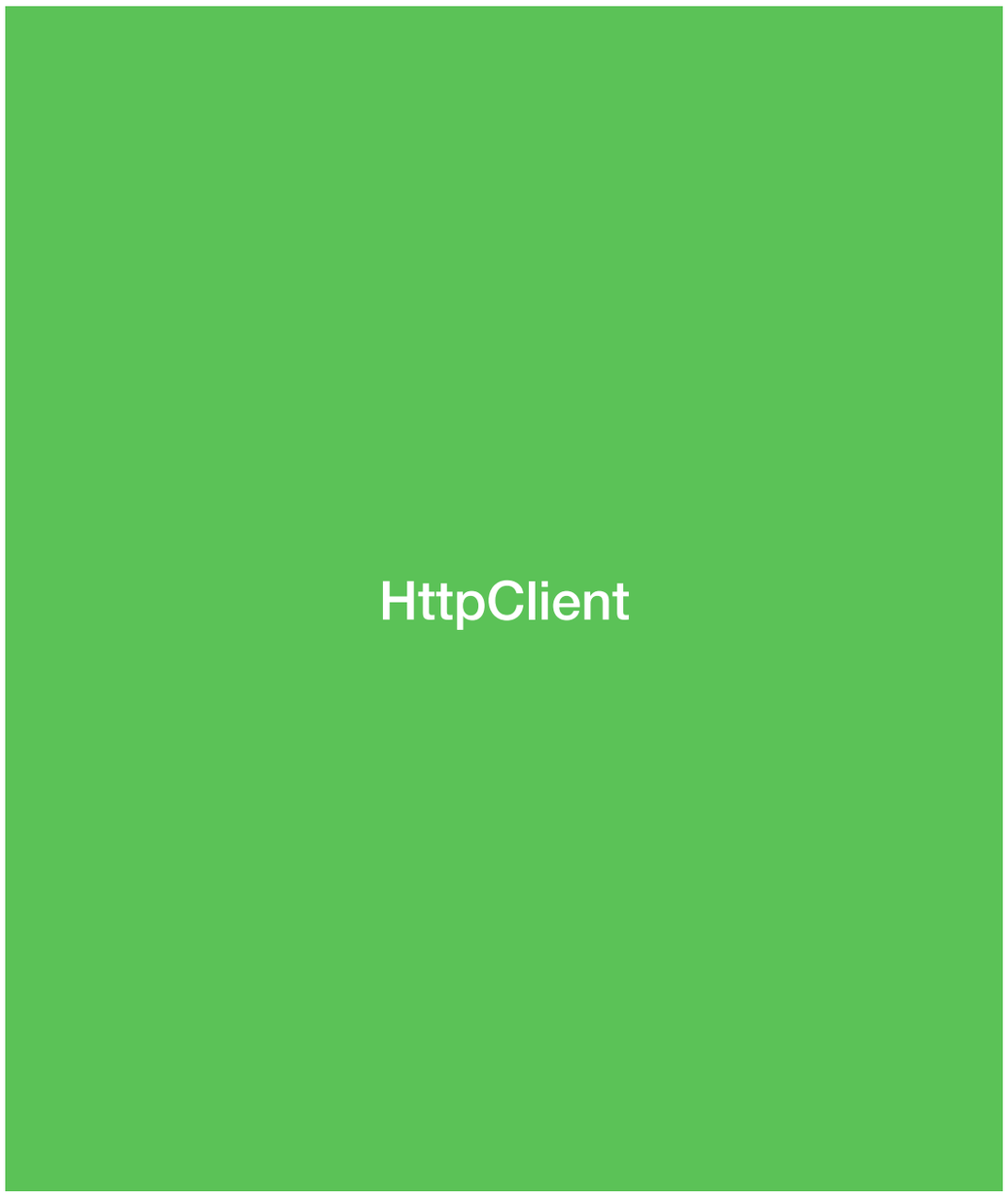
Кусочек сетевого слоя



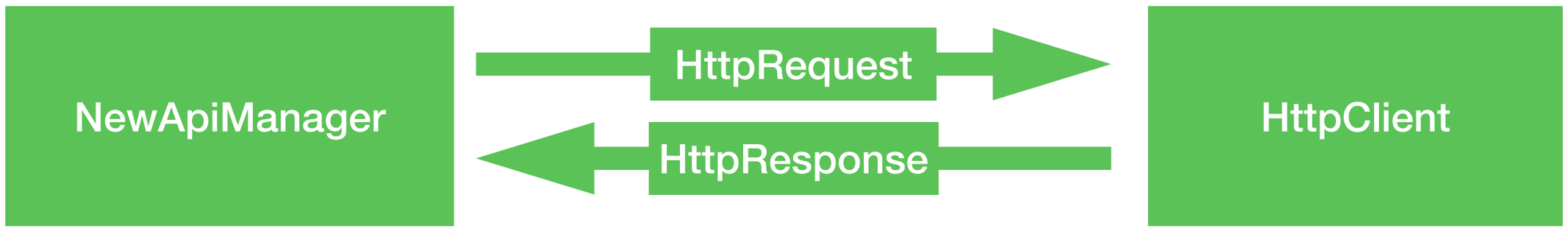
Кусочек сетевого слоя



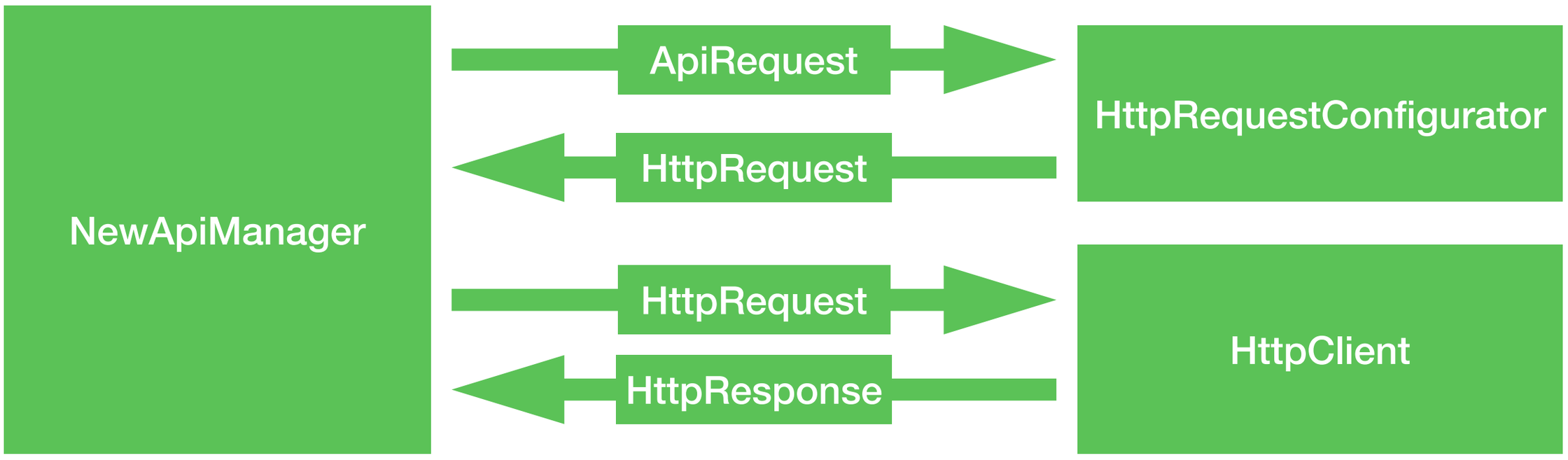
Кусочек сетевого слоя



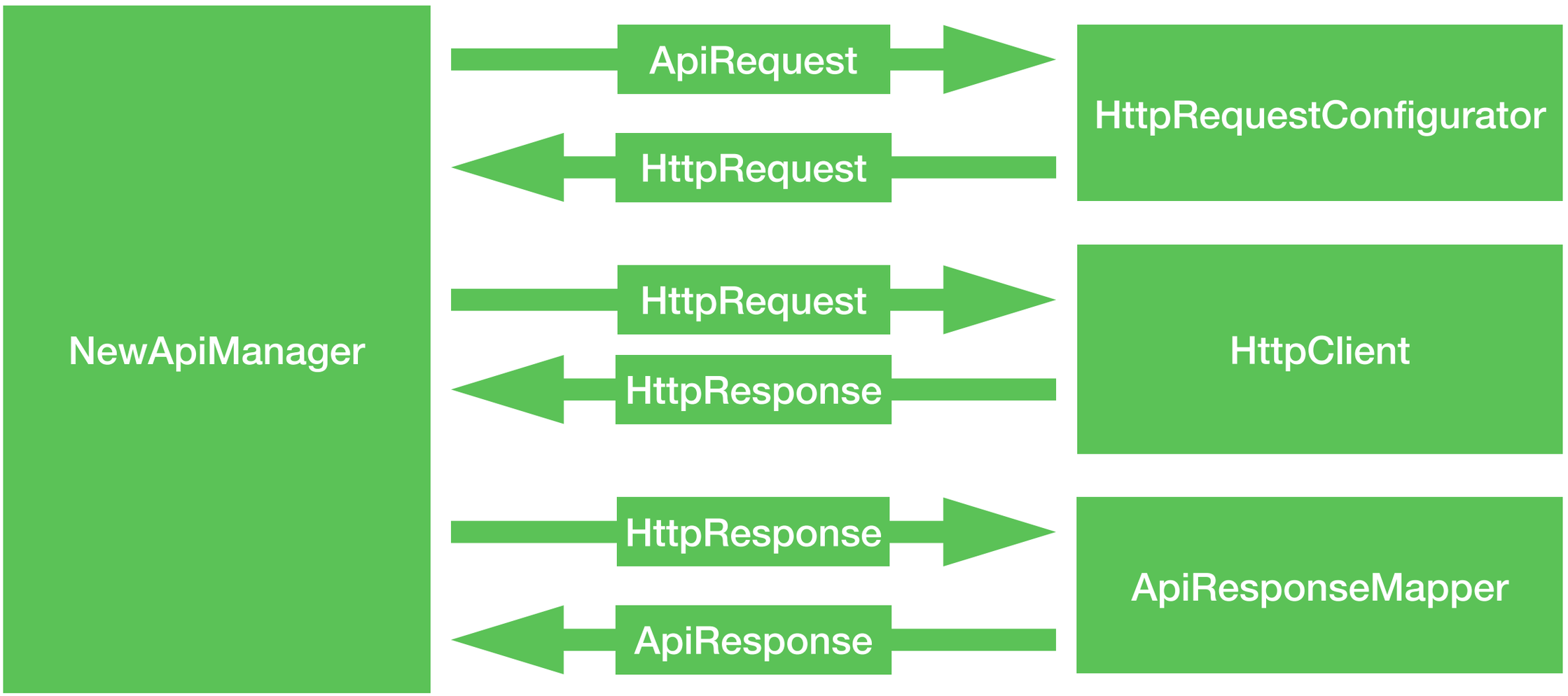
Кусочек сетевого слоя



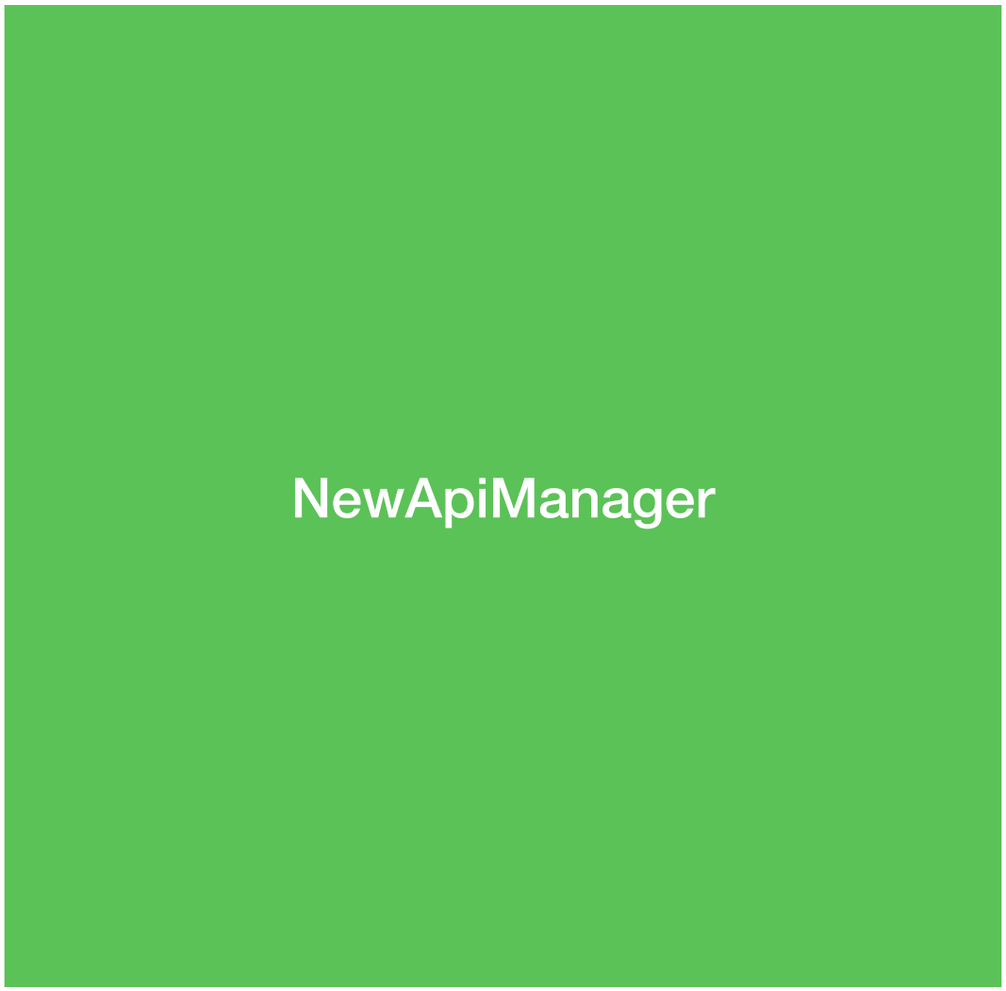
Кусочек сетевого слоя



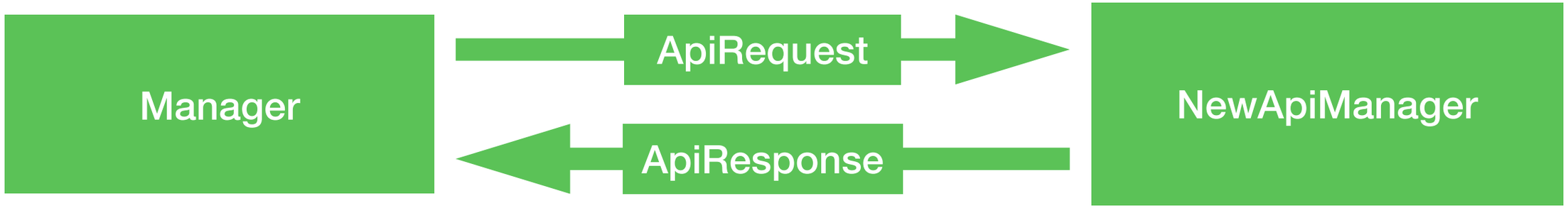
Кусочек сетевого слоя



Кусочек сетевого слоя



Кусочек сетевого слоя



Кусочек сетевого слоя

```
public struct HttpRequest {

    typealias Headers = [String: String]
    typealias Parameters = [String: Any]

    let method: HttpMethod
    let contentType: HttpContentType
    let headers: Headers
    let parameters: Parameters
    let url: URL
}

public struct ApiRequest {

    public let host: ApiHost
    public let method: HttpMethod
    public let contentType: HttpContentType
    public let path: String
    public let parameters: [String: Any]
    public let headers: [String: String]
    public let needsToken: Bool
}
```

Кусочек сетевого слоя

```
public struct HttpRequest {

    typealias Headers = [String: String]
    typealias Parameters = [String: Any]

    let method: HttpMethod
    let contentType: HttpContentType
    let headers: Headers
    let parameters: Parameters
    let url: URL
}

public struct ApiRequest {

    public let host: ApiHost
    public let method: HttpMethod
    public let contentType: HttpContentType
    public let path: String
    public let parameters: [String: Any]
    public let headers: [String: String]
    public let needsToken: Bool
}
```

Кусочек сетевого слоя

```
public struct HttpRequest {

    typealias Headers = [String: String]
    typealias Parameters = [String: Any]

    let method: HttpMethod
    let contentType: HttpContentType
    let headers: Headers
    let parameters: Parameters
    let url: URL
}

public struct ApiRequest {

    public let host: ApiHost
    public let method: HttpMethod
    public let contentType: HttpContentType
    public let path: String
    public let parameters: [String: Any]
    public let headers: [String: String]
    public let needsToken: Bool
}
```

Кусочек сетевого слоя

```
public struct HttpRequest {

    typealias Headers = [String: String]
    typealias Parameters = [String: Any]

    let method: HttpMethod
    let contentType: HttpContentType
    let headers: Headers
    let parameters: Parameters
    let url: URL
}

public struct ApiRequest {

    public let host: ApiHost
    public let method: HttpMethod
    public let contentType: HttpContentType
    public let path: String
    public let parameters: [String: Any]
    public let headers: [String: String]
    public let needsToken: Bool
}
```

Кусочек сетевого слоя

```
public struct HttpRequest {

    typealias Headers = [String: String]
    typealias Parameters = [String: Any]

    let method: HttpMethod
    let contentType: HttpContentType
    let headers: Headers
    let parameters: Parameters
    let url: URL
}

public struct ApiRequest {

    public let host: ApiHost
    public let method: HttpMethod
    public let contentType: HttpContentType
    public let path: String
    public let parameters: [String: Any]
    public let headers: [String: String]
    public let needsToken: Bool
}
```

Отличия от старого

Старый сетевой слой

- 16 Objective-C файлов + 2 Swift extension
- DCApiManager на 600 строк
- DCApiManagerHelper на 300 строк
- DCErrorHandler на 300 строк
- AFNetworking
- ~170 методов Api

~~Старый~~ новый сетевой слой

- 16 Objective-C файлов + 2 Swift extension
- DCApiManager на 600 строк
- DCApiManagerHelper на 300 строк
- DCErrorHandler на 300 строк
- AFNetworking
- ~170 методов Api

~~Старый~~ новый сетевой слой

- ~~16 Objective-C файлов + 2 Swift extension~~
  67 Swift файлов

- DCApiManager на 600 строк

- DCApiManagerHelper на 300 строк

- DCErrorHandler на 300 строк

- AFNetworking

- ~170 методов Api

~~Старый~~ новый сетевой слой

- ~~16 Objective-C файлов + 2 Swift extension~~
  67 Swift файлов

- ~~DCApiManager на 600 строк~~
  ~~DCApiManagerHelper на 300 строк~~
  ~~DCErrorHandler на 300 строк~~
  2 файла по 250 строк
  7 файлов меньше 200 строк
  Остальные меньше 100 строк

- AFNetworking

- ~170 методов Api

~~Старый~~ новый сетевой слой

- ~~16 Objective-C файлов + 2 Swift extension~~
  67 Swift файлов

- ~~DCApiManager на 600 строк~~
  ~~DCApiManagerHelper на 300 строк~~
  ~~DCErrorHandler на 300 строк~~
  2 файла по 250 строк
  7 файлов меньше 200 строк
  Остальные меньше 100 строк

- ~~AFNetworking~~
  URLSession, закрытый протоколом

- ~170 методов Api

Раскатка

Начинаем раскатывать

И...

Что-то пошло не так

Как думаете что?


Баги?


Крешш?

Что же?

Дима решил уволиться

Кто такой Дима?

Автор нового сетевого слоя!

Казалось новому сетевому слою суждено было стать Legacy до официального релиза, но…

Потому что за него

ВЗЯЛСЯ Я

← помните вот этого паренька?

Шутка, не поэтому

Чем же он хорош?

не Дима, а сетевой слой

но и Дима - красавчик!

S

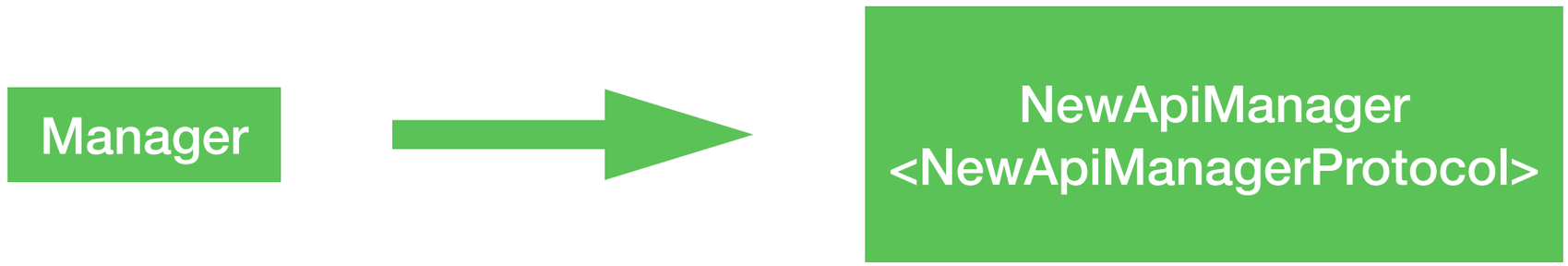
Еще паттерны

1.

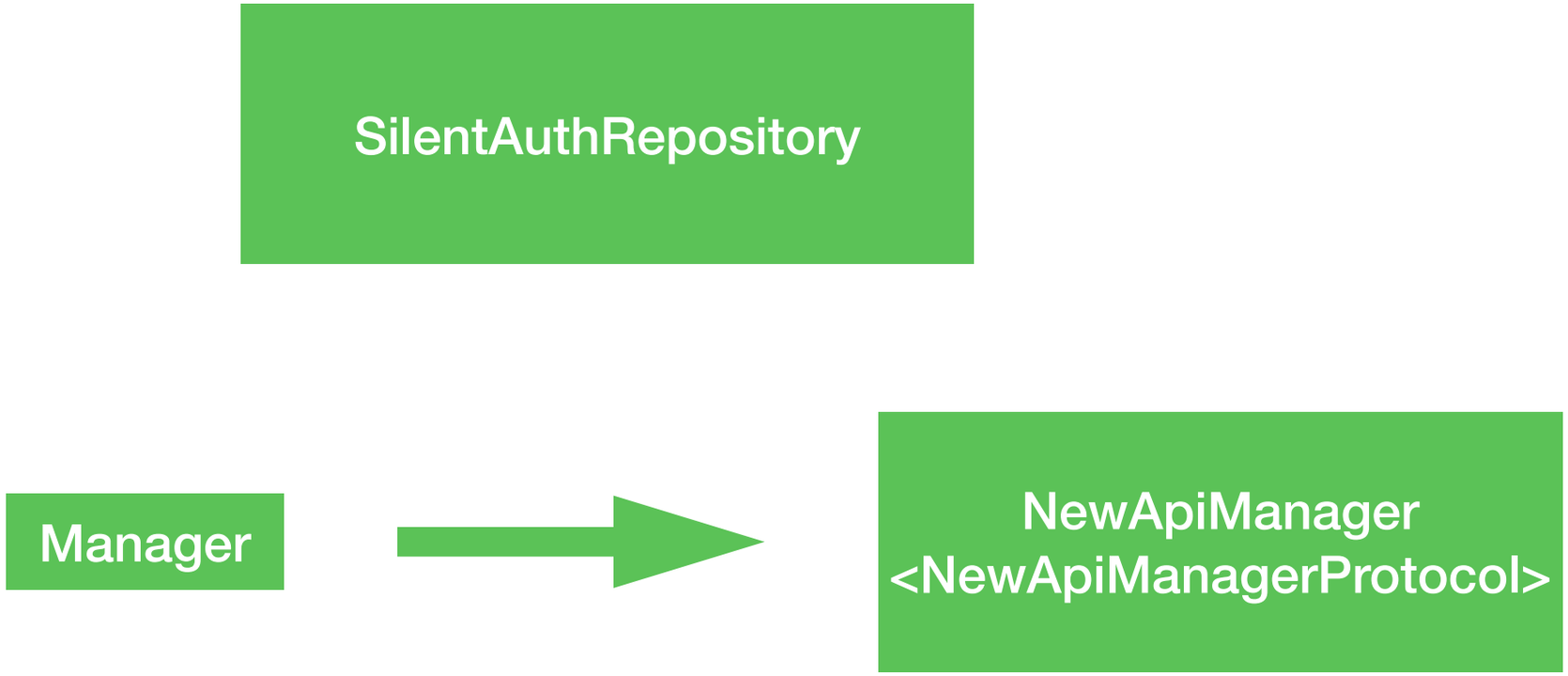
2.

3.

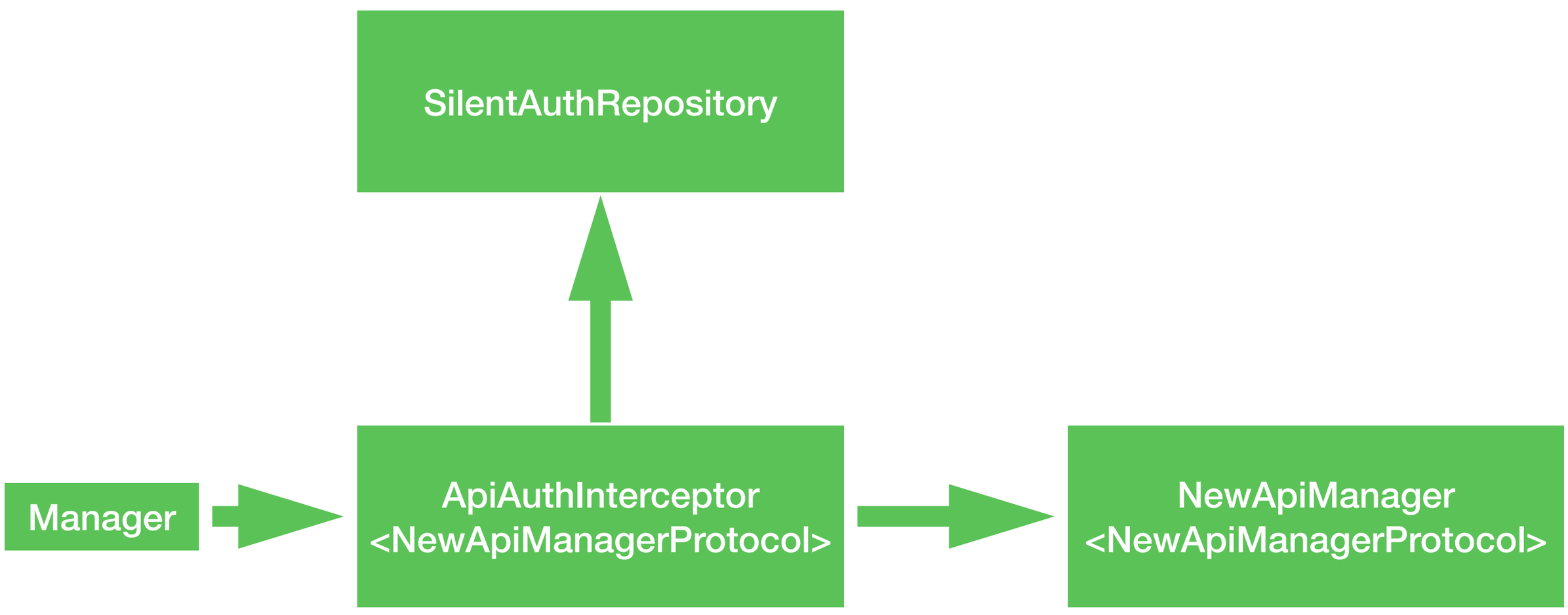
Что за паттерн?



Что за паттерн?



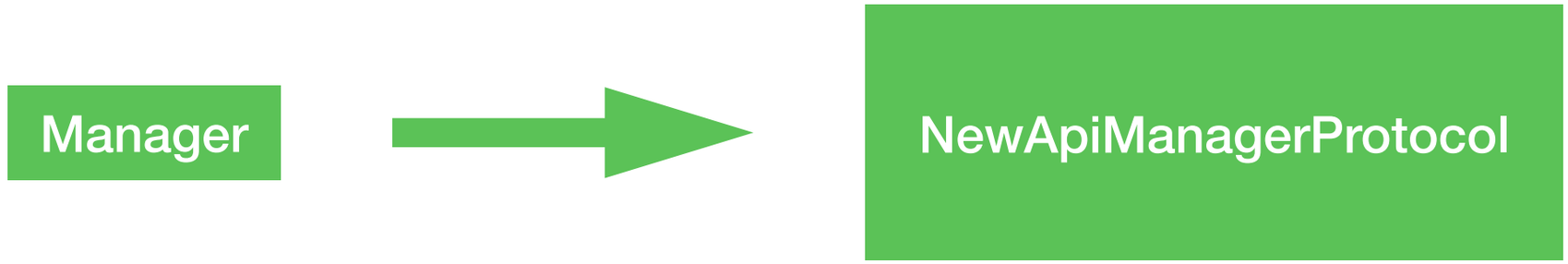
Что за паттерн?



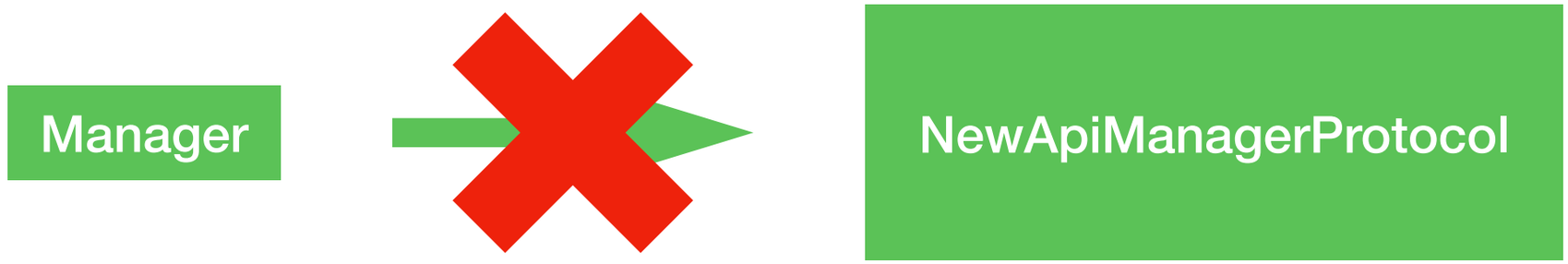
Еще паттерны

1. Декоратор
2.
3.

Что за паттерн?



Что за паттерн?

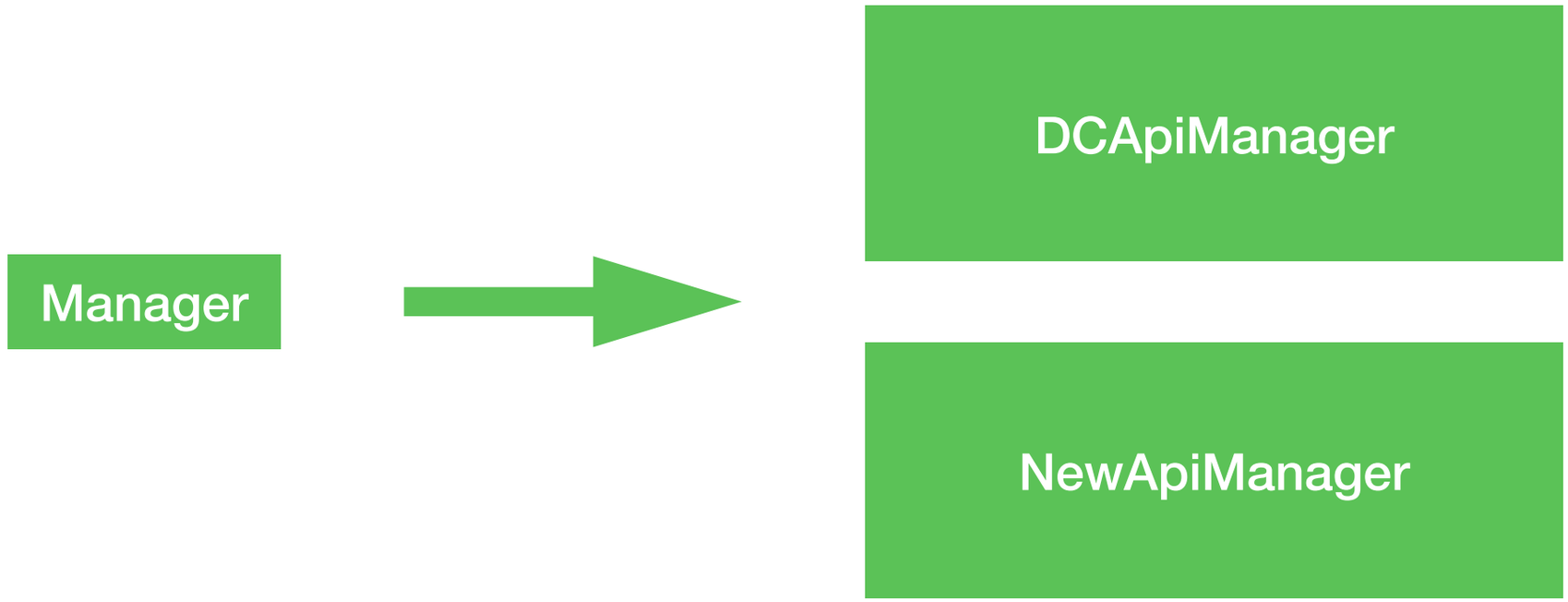


Что за паттерн?

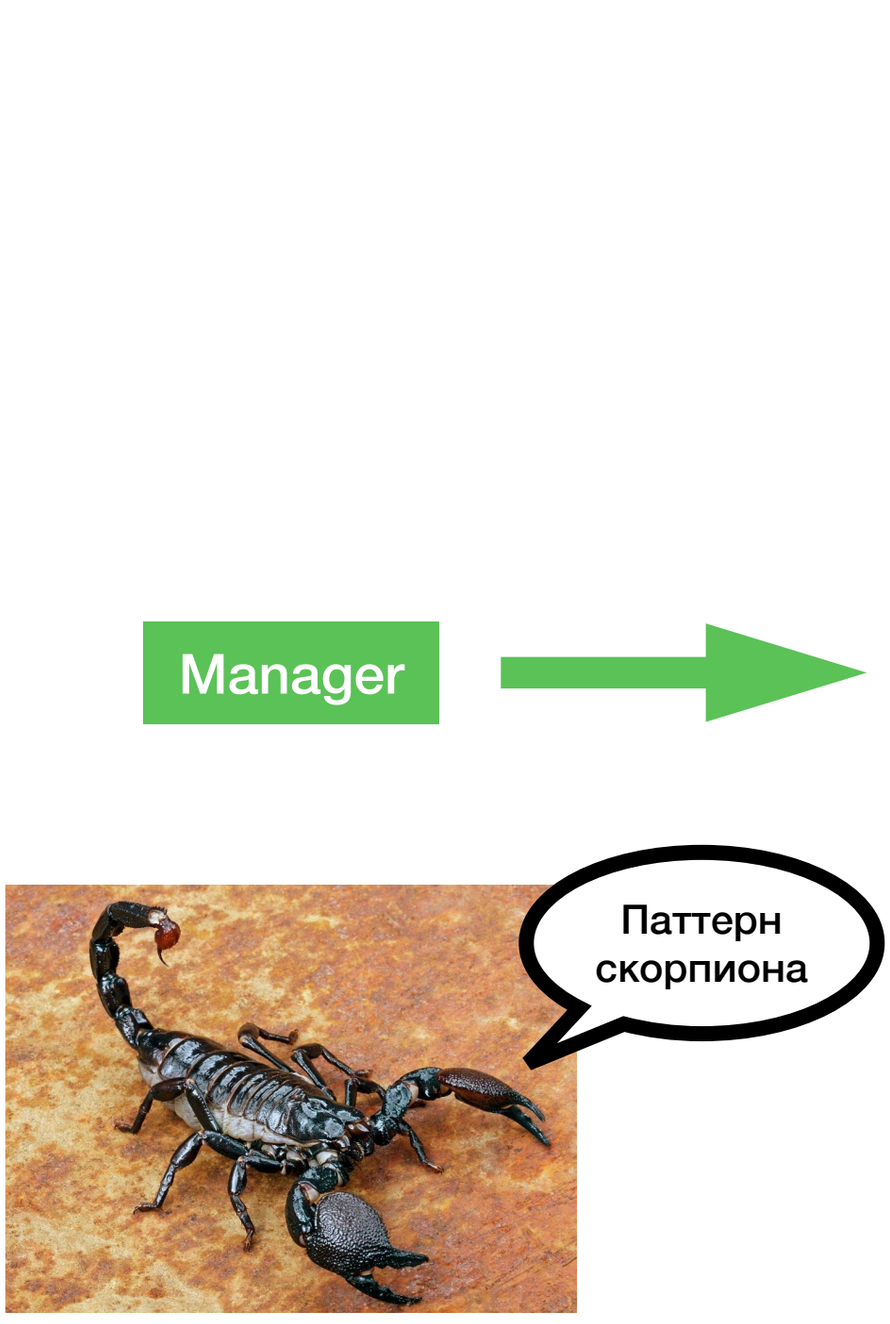
Еще паттерны

1. Декоратор
2. Адаптер
3.

Что за паттерн?



Что за паттерн?



Еще паттерны

1. Декоратор
2. Адаптер
3. Фасад

Баг с разлогином, помните?

Решение

```
final class ApiAuthInterceptor: NewApiManagerProtocol {

    private let stateMachine = ApiAuthStateMachine()
    private var externalRequests = [ApiPendingRequest]()
    private var internalRequests = [ApiPendingRequest]()

    ...
}
```

Решение

```
final class ApiAuthInterceptor: NewApiManagerProtocol {

    private let stateMachine = ApiAuthStateMachine()
    private var externalRequests = [ApiPendingRequest]()
    private var internalRequests = [ApiPendingRequest]()

    ...
}
```

Решение

```
final class ApiAuthInterceptor: NewApiManagerProtocol {

    private let stateMachine = ApiAuthStateMachine()
    private var externalRequests = [ApiPendingRequest]()
    private var internalRequests = [ApiPendingRequest]()

    ...
}
```

Решение

```
final class ApiAuthInterceptor: NewApiManagerProtocol {

    private let stateMachine = ApiAuthStateMachine()
    private var externalRequests = [ApiPendingRequest]()
    private var internalRequests = [ApiPendingRequest]()

    ...
}
```

Решение

```
private struct ApiPendingRequest {

    let request: ApiRequest
    let error: ApiError?
    let completion: ((ApiRequestResult) -> Void)

    func fail(with error: Error) {
        completion(.failure(self.error ?? error))
    }

}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {
    if stateMachine.isExecuting && request.needsToken {
        let pendingRequest = ApiPendingRequest(
            request: request,
            error: nil,
            completion: completion
        )
        externalRequests.append(pendingRequest)
        return
    }

    ...

}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {
    if stateMachine.isExecuting && request.needsToken {
        let pendingRequest = ApiPendingRequest(
            request: request,
            error: nil,
            completion: completion
        )
        externalRequests.append(pendingRequest)
        return
    }

    ...

}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {
    if stateMachine.isExecuting && request.needsToken {
        let pendingRequest = ApiPendingRequest(
            request: request,
            error: nil,
            completion: completion
        )
        externalRequests.append(pendingRequest)
        return
    }

    ...

}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {
    if stateMachine.isExecuting && request.needsToken {
        let pendingRequest = ApiPendingRequest(
            request: request,
            error: nil,
            completion: completion
        )
        externalRequests.append(pendingRequest)
        return
    }

    ...

}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {

    ...

    apiManager.perform(request: request, completion: {
        [weak self] result in
        do {
            let response = try result.get()
            completion(.success(response))
        } catch ApiError.authorization(let error) {
            self?.handleAuthorizationError(
                error,
                request: request,
                completion: completion
            )
        } catch {
            completion(.failure(error))
        }
    })
}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {

    ...

    apiManager.perform(request: request, completion: {
        [weak self] result in
        do {
            let response = try result.get()
            completion(.success(response))
        } catch ApiError.authorization(let error) {
            self?.handleAuthorizationError(
                error,
                request: request,
                completion: completion
            )
        } catch {
            completion(.failure(error))
        }
    })
}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {

    ...

    apiManager.perform(request: request, completion: {
        [weak self] result in
        do {
            let response = try result.get()
            completion(.success(response))
        } catch ApiError.authorization(let error) {
            self?.handleAuthorizationError(
                error,
                request: request,
                completion: completion
            )
        } catch {
            completion(.failure(error))
        }
    })
}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {

    ...

    apiManager.perform(request: request, completion: {
        [weak self] result in
        do {
            let response = try result.get()
            completion(.success(response))
        } catch ApiError.authorization(let error) {
            self?.handleAuthorizationError(
                error,
                request: request,
                completion: completion
            )
        } catch {
            completion(.failure(error))
        }
    })
}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {

    ...

    apiManager.perform(request: request, completion: {
        [weak self] result in
        do {
            let response = try result.get()
            completion(.success(response))
        } catch ApiError.authorization(let error) {
            self?.handleAuthorizationError(
                error,
                request: request,
                completion: completion
            )
        } catch {
            completion(.failure(error))
        }
    })
}
```

Решение

```
func perform(request: ApiRequest, completion: @escaping Completion) {

    ...

    apiManager.perform(request: request, completion: {
        [weak self] result in
        do {
            let response = try result.get()
            completion(.success(response))
        } catch ApiError.authorization(let error) {
            self?.handleAuthorizationError(
                error,
                request: request,
                completion: completion
            )
        } catch {
            completion(.failure(error))
        }
    })
}
```

Решение

```
private func handleAuthorizationError(
    _ error: ApiError.Authorization,
    request: ApiRequest,
    completion: @escaping Completion
) {
    let pendingRequest = ApiPendingRequest(
        request: request,
        error: .authorization(error),
        completion: completion
    )
    internalRequests.append(pendingRequest)
    guard !stateMachine.isExecuting else {
        return
    }
    stateMachine.process(event: .startSilentAuth)
}
```

Решение

```
private func handleAuthorizationError(
    _ error: ApiError.Authorization,
    request: ApiRequest,
    completion: @escaping Completion
) {
    let pendingRequest = ApiPendingRequest(
        request: request,
        error: .authorization(error),
        completion: completion
    )
    internalRequests.append(pendingRequest)
    guard !stateMachine.isExecuting else {
        return
    }
    stateMachine.process(event: .startSilentAuth)
}
```

Решение

```
private func handleAuthorizationError(
    _ error: ApiError.Authorization,
    request: ApiRequest,
    completion: @escaping Completion
) {
    let pendingRequest = ApiPendingRequest(
        request: request,
        error: .authorization(error),
        completion: completion
    )
    internalRequests.append(pendingRequest)
    guard !stateMachine.isExecuting else {
        return
    }
    stateMachine.process(event: .startSilentAuth)
}
```

Решение

```
private func handleAuthorizationError(
    _ error: ApiError.Authorization,
    request: ApiRequest,
    completion: @escaping Completion
) {
    let pendingRequest = ApiPendingRequest(
        request: request,
        error: .authorization(error),
        completion: completion
    )
    internalRequests.append(pendingRequest)
    guard !stateMachine.isExecuting else {
        return
    }
    stateMachine.process(event: .startSilentAuth)
}
```

Решение

```
private func handleAuthorizationError(
    _ error: ApiError.Authorization,
    request: ApiRequest,
    completion: @escaping Completion
) {
    let pendingRequest = ApiPendingRequest(
        request: request,
        error: .authorization(error),
        completion: completion
    )
    internalRequests.append(pendingRequest)
    guard !stateMachine.isExecuting else {
        return
    }
    stateMachine.process(event: .startSilentAuth)
}
```

Решение

```
private func handleIdleState(
    with result: ApiAuthEvent.Result,
    previousState: ApiAuthState
) {
    let requestsToInvoke = internalRequests + externalRequests
    internalRequests.removeAll()
    externalRequests.removeAll()
    switch result {
    case .success:
        requestsToInvoke.forEach(perform)
    case .failure(let error):
        requestsToInvoke.forEach({ $0.fail(with: error) })
    }
}
```

Решение

```
private func handleIdleState(
    with result: ApiAuthEvent.Result,
    previousState: ApiAuthState
) {
    let requestsToInvoke = internalRequests + externalRequests
    internalRequests.removeAll()
    externalRequests.removeAll()
    switch result {
    case .success:
        requestsToInvoke.forEach(perform)
    case .failure(let error):
        requestsToInvoke.forEach({ $0.fail(with: error) })
    }
}
```

Решение

```
private func handleIdleState(
    with result: ApiAuthEvent.Result,
    previousState: ApiAuthState
) {
    let requestsToInvoke = internalRequests + externalRequests
    internalRequests.removeAll()
    externalRequests.removeAll()
    switch result {
    case .success:
        requestsToInvoke.forEach(perform)
    case .failure(let error):
        requestsToInvoke.forEach({ $0.fail(with: error) })
    }
}
```

Решение

```
private func handleIdleState(
    with result: ApiAuthEvent.Result,
    previousState: ApiAuthState
) {
    let requestsToInvoke = internalRequests + externalRequests
    internalRequests.removeAll()
    externalRequests.removeAll()
    switch result {
    case .success:
        requestsToInvoke.forEach(perform)
    case .failure(let error):
        requestsToInvoke.forEach({ $0.fail(with: error) })
    }
}
```

Решение

```
private func handleIdleState(
    with result: ApiAuthEvent.Result,
    previousState: ApiAuthState
) {
    let requestsToInvoke = internalRequests + externalRequests
    internalRequests.removeAll()
    externalRequests.removeAll()
    switch result {
    case .success:
        requestsToInvoke.forEach(perform)
    case .failure(let error):
        requestsToInvoke.forEach({ $0.fail(with: error) })
    }
}
```

Решение

- Новый запрос не выполняется, если стейт машина работает
- Ответ 401 не обрабатывается, если стейт машина работает
- Стейт машина потокобезопасна

Решение

- Новый запрос не выполняется, если стейт машина работает
- Ответ 401 не обрабатывается, если стейт машина работает
- Стейт машина потокобезопасна

Решение

- Новый запрос не выполняется, если стейт машина работает

- Ответ 401 не обрабатывается, если стейт машина работает

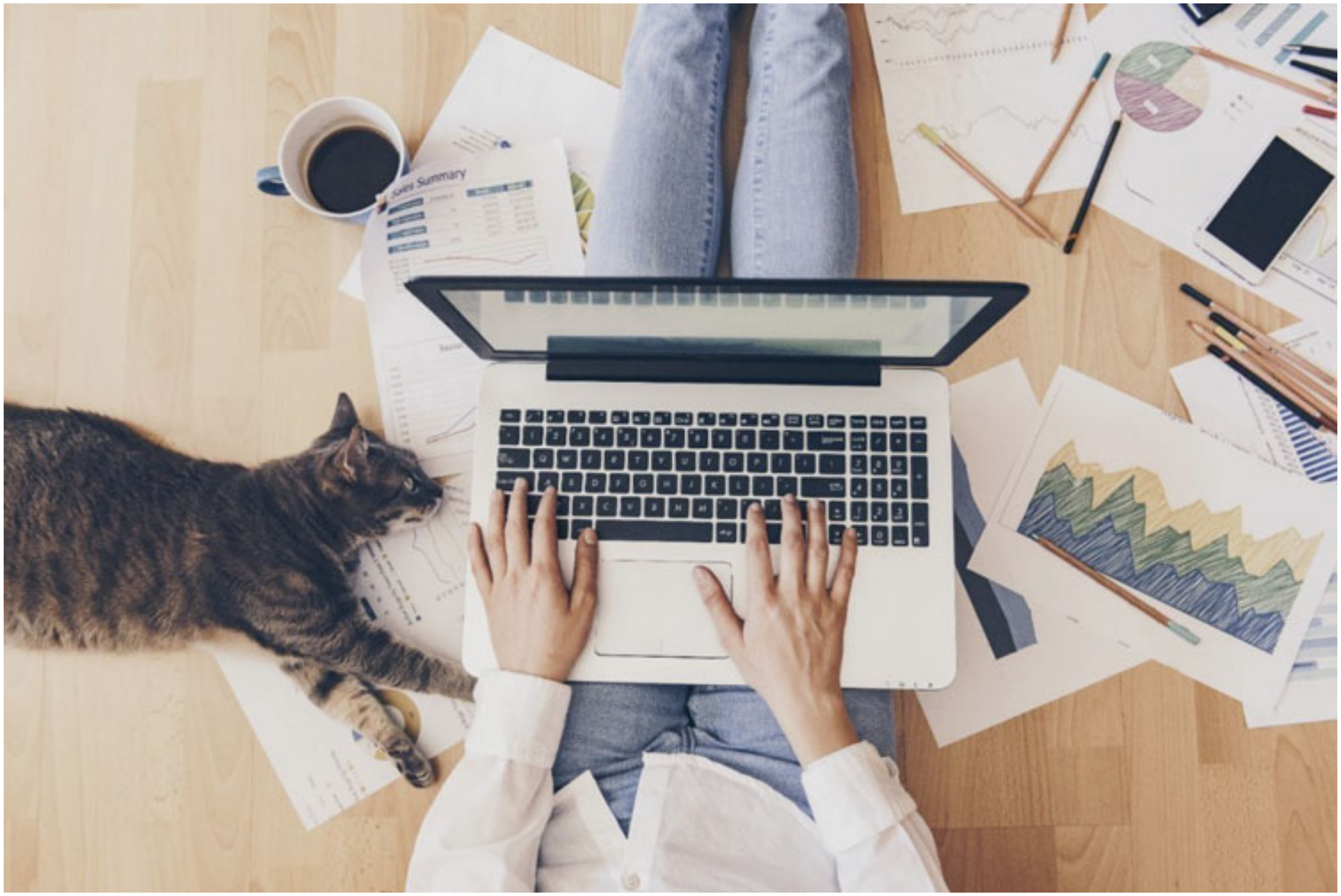
- Стейт машина потокобезопасна

О чем я расскажу

- Интересная история
- ~~Причины переписывания сетевого слоя~~ 
- ~~Хорошие практики~~ 

И снова раскатка

это моя жена →



↑

это я

Раскатка на 50%

Ошибки тихой авторизации

Как мы узнали?

- Firebase Crashlytics
- Grafana
- Алерты в чат

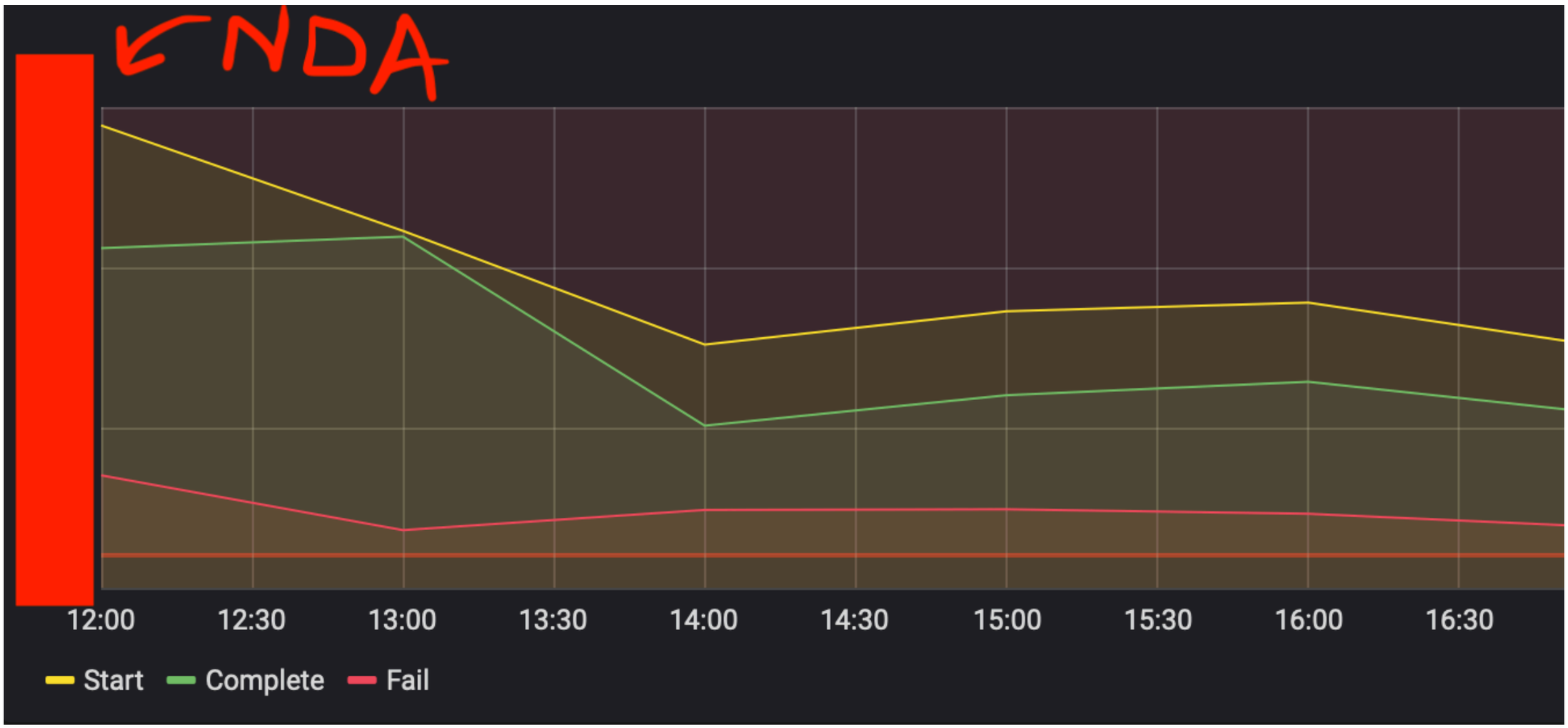
Как мы узнали?

- Firebase Crashlytics
- Grafana
- Алерты в чат

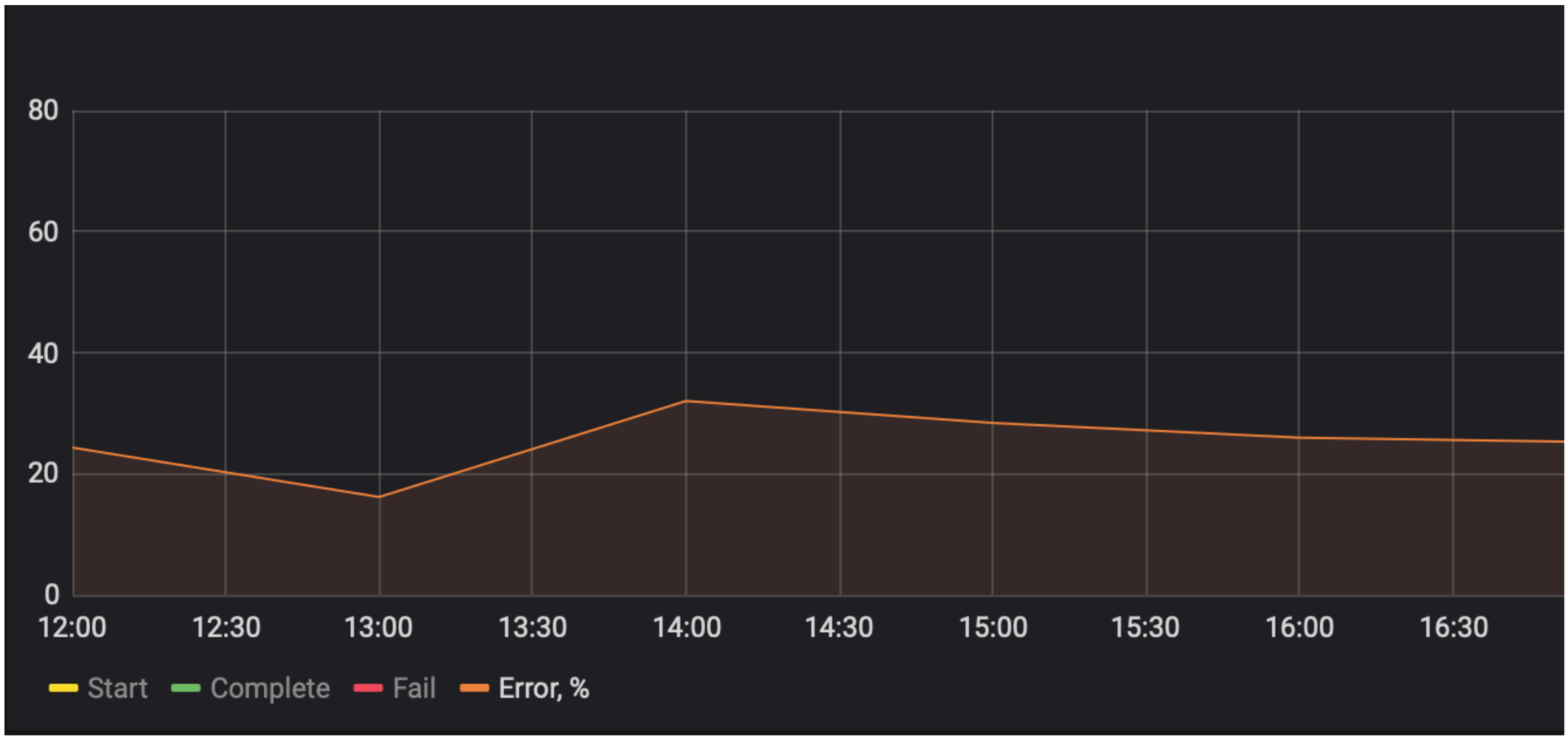
Как мы узнали?

- Firebase Crashlytics
- Grafana
- Алерты в чат

Grafana


NDA
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
Start
Complete
Fail

Grafana


80
60
40
20
0
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
Start
Complete
Fail
Error, %

Причина

- Баг в логике ApiAuthInterceptor

Решение

- Написать тесты на проверку реакций на все ошибки
- Один из них будет красным
- Поправить логику
- Убедиться что все тесты зеленые

Решение

- Написать тесты на проверку реакций на все ошибки
- Один из них будет красным
- Поправить логику
- Убедиться что все тесты зеленые

Решение

- Написать тесты на проверку реакций на все ошибки
- Один из них будет красным
- Поправить логику
- Убедиться что все тесты зеленые

Решение

- Написать тесты на проверку реакций на все ошибки
- Один из них будет красным
- Поправить логику
- Убедиться что все тесты зеленые

Раскатка на 100%

Краш на старте...

Информация

- Креш в main во время стартовых запросов
- Появляется только в версиях с новым сетевым слоем
- Не воспроизводится

Информация

- Креш в main во время стартовых запросов
- Появляется только в версиях с новым сетевым слоем
- Не воспроизводится

Информация

- Креш в main во время стартовых запросов
- Появляется только в версиях с новым сетевым слоем
- Не воспроизводится

Поговорили про хорошие практики, поговорим и про...

Я этим не горжусь

но рассказать хочу

Я этим не горжусь

но рассказать хочу

```
- (void)requestWithApiConfig:(DCApiConfig * _Nonnull)config {
    @synchronized (apiManagerHelper) {
        ...
    }
}
```

Я этим не горжусь

но рассказать хочу

```
- (void)requestWithApiConfig:(DCApiConfig * _Nonnull)config {
    @synchronized (apiManagerHelper) {
        ...
    }
}
```

Я этим не горжусь

но рассказать хочу

```objc
- (void)requestWithApiConfig:(DCApiConfig * _Nonnull)config {
    @synchronized (apiManagerHelper) {
        ...
    }
}
```

```swift
func requestWithApiConfig(_ config: DCApiConfig) {
    ...
}
```

Я этим не горжусь

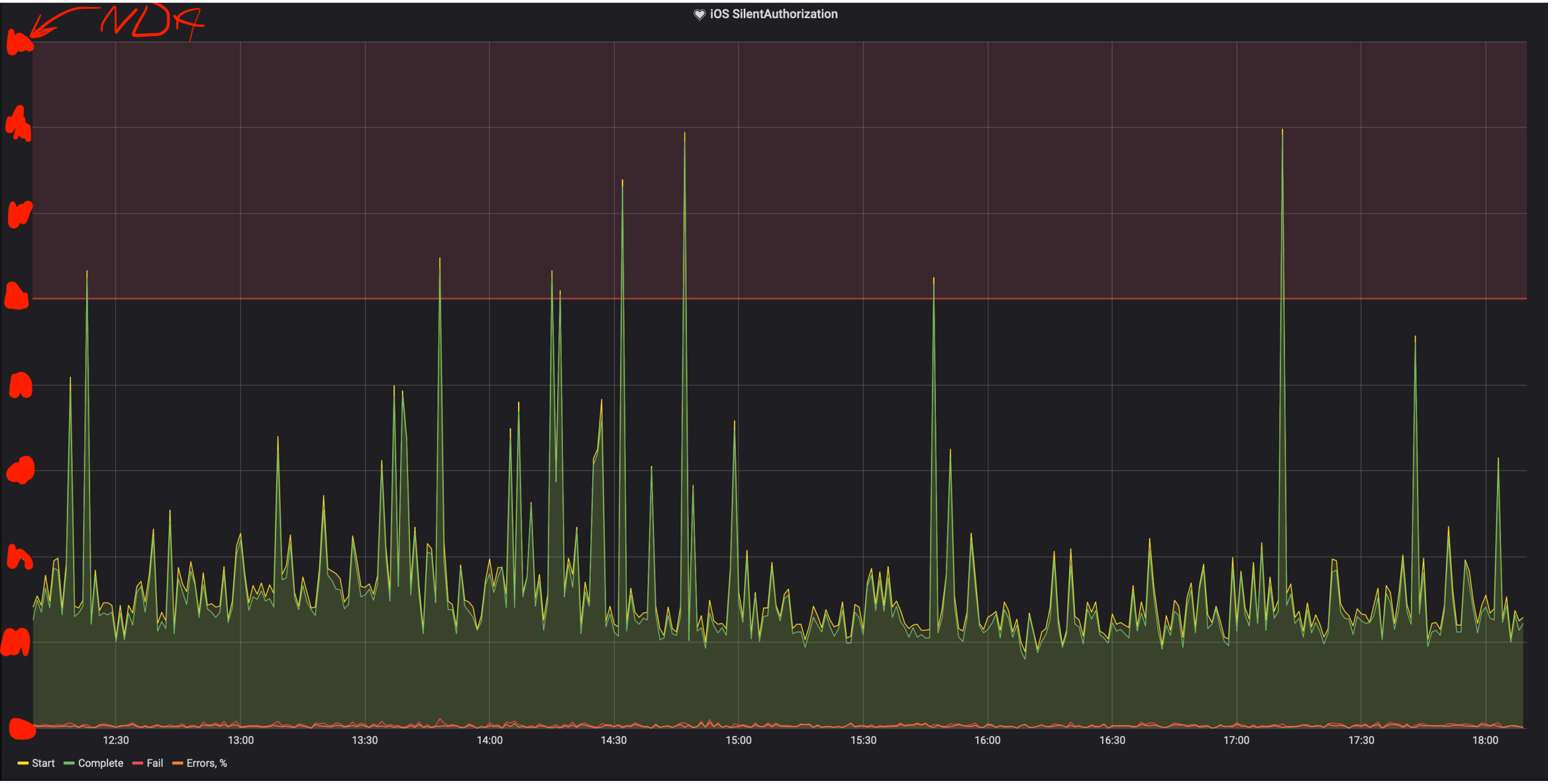
но рассказать хочу

```objc
- (void)requestWithApiConfig:(DCApiConfig * _Nonnull)config {
    @synchronized (apiManagerHelper) {
        ...
    }
}
```

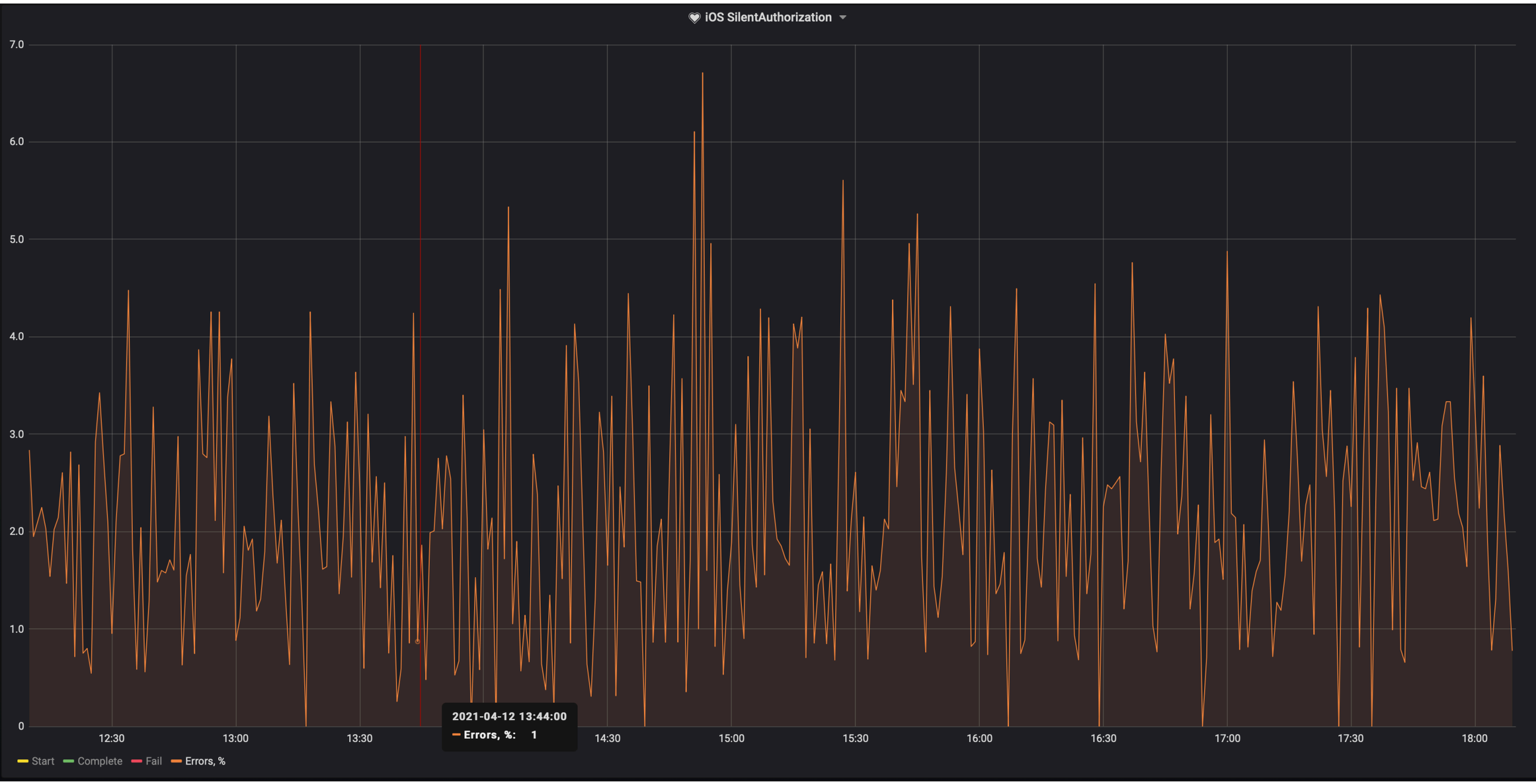
```swift
let lock = NSRecursiveLock()

func requestWithApiConfig(_ config: DCApiConfig) {
    lock.lock()
    ...
    lock.unlock()
}
```

УРА!!!

Ну кайф же!


NDA
iOS SilentAuthorization
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
17:00
17:30
18:00
Start
Complete
Fail
Errors, %

Ну кайф же!


iOS SilentAuthorization
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0
12:30
13:00
13:30
14:30
15:00
15:30
16:00
16:30
17:00
17:30
18:00
2021-04-12 13:44:00
Errors, %: 1
Start
Complete
Fail
Errors, %

О чем я расскажу

- ~~Интересная история~~ 
- ~~Причины переписывания сетевого слоя~~ 
- ~~Хорошие практики~~ 

Дальнейшие планы

- Выпиливание старого сетевого слоя
- Оптимизация Codable решения
- Выпиливание Obj-C моделей

Выпиливание старого сетевого слоя

Выпиливание старого сетевого слоя

```
@objc
public protocol ApiManagerProtocol: AnyObject {

    func requestWithApiConfig(_ config: DCApiConfig)

    ...

}
```

Выпиливание старого сетевого слоя

```
@objc
public protocol ApiManagerProtocol: AnyObject {

    func requestWithApiConfig(_ config: DCApiConfig)

    ...

}


public typealias ApiRequestResult = (Result<ApiResponse, Error>)

public protocol NewApiManagerProtocol: AnyObject {
    typealias Completion = (ApiRequestResult) -> Void
    func perform(request: ApiRequest, completion: @escaping Completion)
}
```

Выпиливание старого сетевого слоя

```
@objc
public protocol ApiManagerProtocol: AnyObject {

    func requestWithApiConfig(_ config: DCApiConfig)
        ...
}


public typealias ApiRequestResult = (Result<ApiResponse, Error>)

public protocol NewApiManagerProtocol: AnyObject {
    typealias Completion = (ApiRequestResult) -> Void
    func perform(request: ApiRequest, completion: @escaping Completion)
}
```

Выпиливание старого сетевого слоя

- ApiManagerFacade
- ApiManagerAdapter

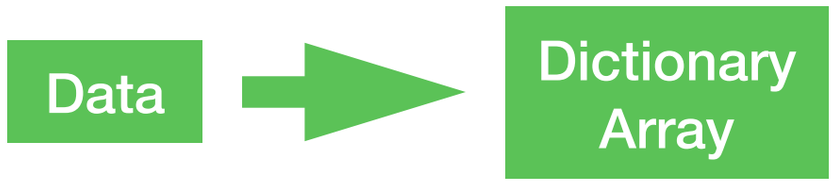
Дальнейшие планы

- Выпиливание старого сетевого слоя
- Оптимизация Codable решения
- Выпиливание Obj-C моделей

Оптимизация Codable решения

Data

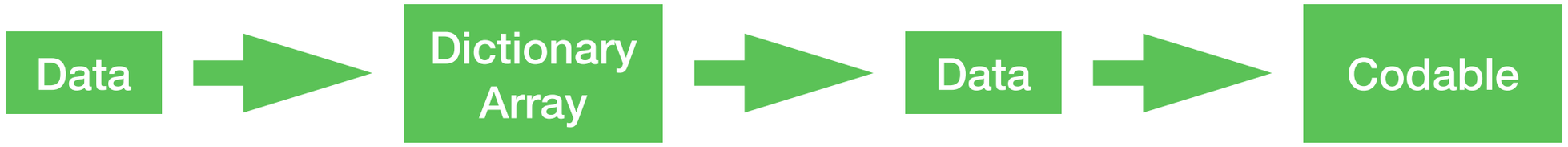
Оптимизация Codable решения



Оптимизация Codable решения

Оптимизация Codable решения

Оптимизация Codable решения



Дальнейшие планы

- Выпиливание старого сетевого слоя
- Оптимизация Codable решения
- Выпиливание Obj-C моделей

Выводы

- Не бойтесь переписывать сетевой слой
- Замените его для всех запросов сразу
- Покройте тестами
- Подстрахуйтесь от ошибок в проде

Выводы

- Не бойтесь переписывать сетевой слой
- Замените его для всех запросов сразу
- Покройте тестами
- Подстрахуйтесь от ошибок в проде

Выводы

- Не бойтесь переписывать сетевой слой
- Замените его для всех запросов сразу
- Покройте тестами
- Подстрахуйтесь от ошибок в проде

Выводы

- Не бойтесь переписывать сетевой слой
- Замените его для всех запросов сразу
- Покройте тестами
- Подстрахуйтесь от ошибок в проде

Спасибо за внимание!

Преза тут