Урок 11

Аналитика

Статистика и аналитика приложения.

Сбор статистики и аналитика выпущенного приложения

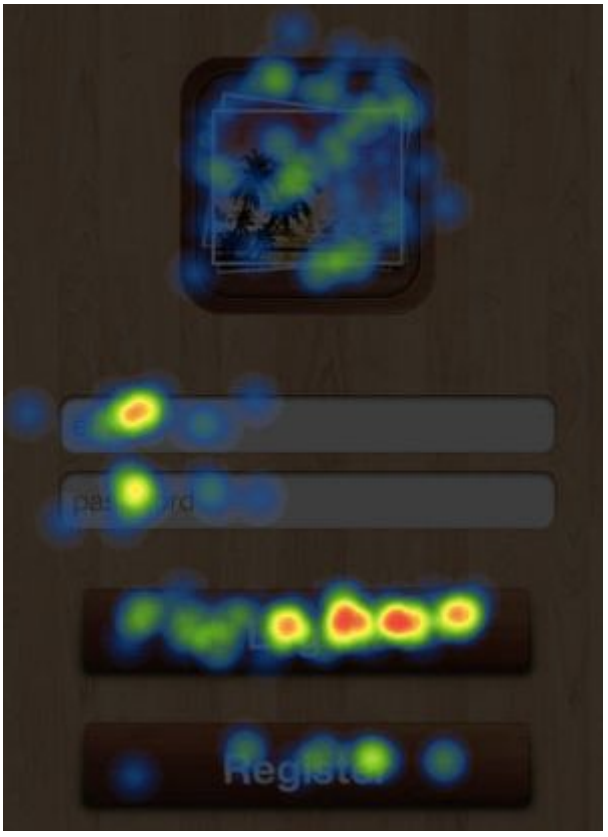
Разработка и продвижение крупного iOS-приложения невозможна без анализа того, что делает пользователь в нем. Сбор подобных данных позволяет принять решение о дальнейшем развитии продукта, эффективности рекламных кампаний.

В основном это маркетинговая информация, но и разработчикам может быть интересна данная статистика. Например, мобильное приложение является еще одной фронт-системой для сервиса. Для него уже написана web-реализация, а для функциональности нужно последовательно выполнить несколько запросов подряд. Но тут появляется наше приложение с медленным мобильным интернетом, и комплекс этих запросов начинает сбоить — не укладываться в тайминги. Используя статистику, можем наблюдать, что пользователи часто используют эту функциональность, а она редко срабатывает. Результат — для мобильного приложения формируется один запрос для обработки функциональности.

В реальном проекте «Мобильный Банк» на основании статистики переходов пользователей внутри приложения было решено отказаться от раздела «Пополнение наличными». Пользователи хоть и знали, где находится этот пункт, не пользовались им. А раздел «Перевод с карты на карту», находящийся глубже, оказался очень популярным, что привело к решению перенести этот пункт на одну из главных страниц.

Но если покрывать приложение статистикой по принципу «когда-нибудь пригодится», она становится бесполезной. Прежде чем включать сбор статистики, подумайте, что вы хотите получить от нее, и выпишите полный список событий, которые необходимо покрыть статистикой.

Существует множество решений для сбора статистики — как платных, так и бесплатных. Они позволяют получать оперативную информацию об использовании приложения, его ежедневной аудитории, путях пользователей в приложении, часто используемых функциях, предоставляют тепловые карты. Последние — скорее приятное дополнение, чем необходимый инструмент, но и они бывают полезны для выявления проблем в интерфейсе. Выглядит тепловая карта примерно так:



Наиболее популярны Crashlytics, Яндекс.Метрика. Рассмотрим пример встраивания Crashlytics от Google в проект.

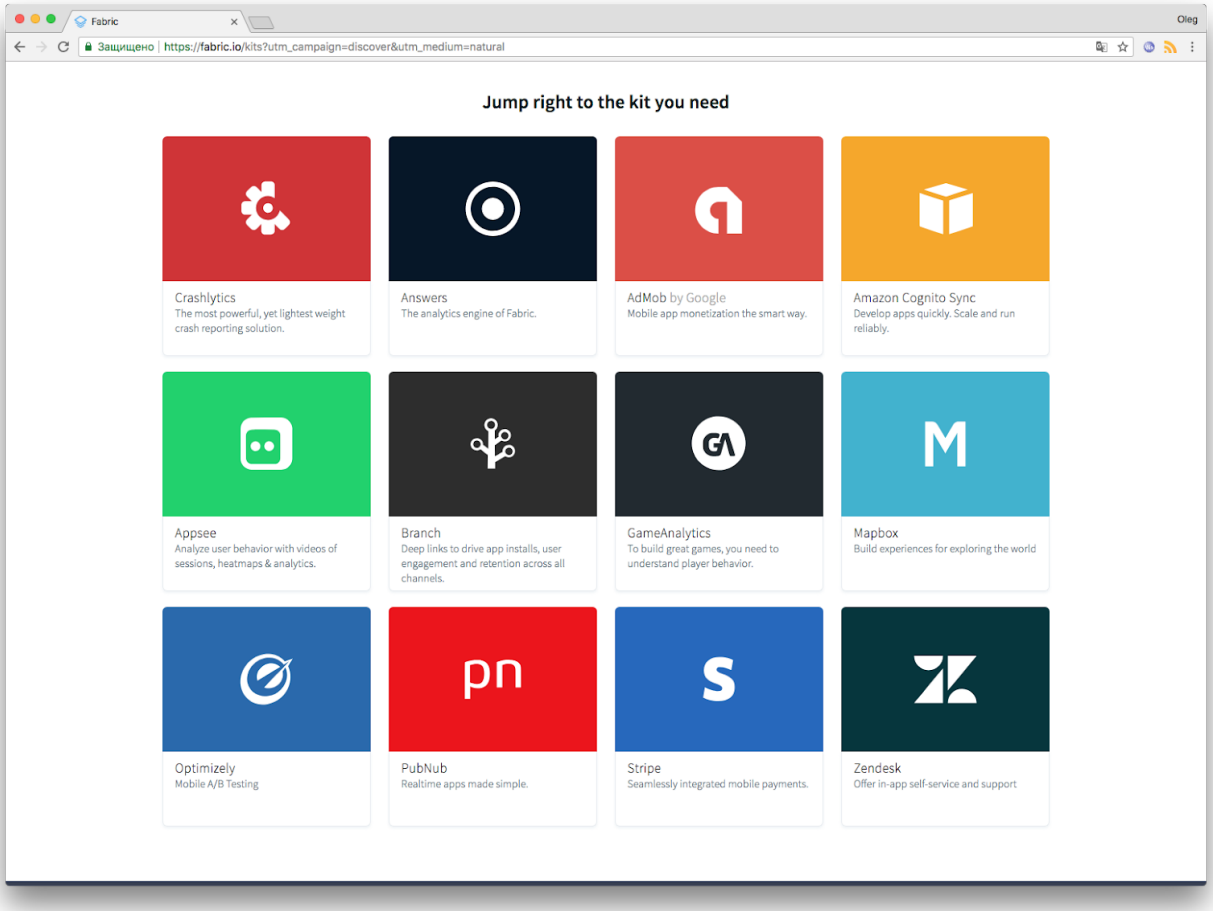
Crashlytics

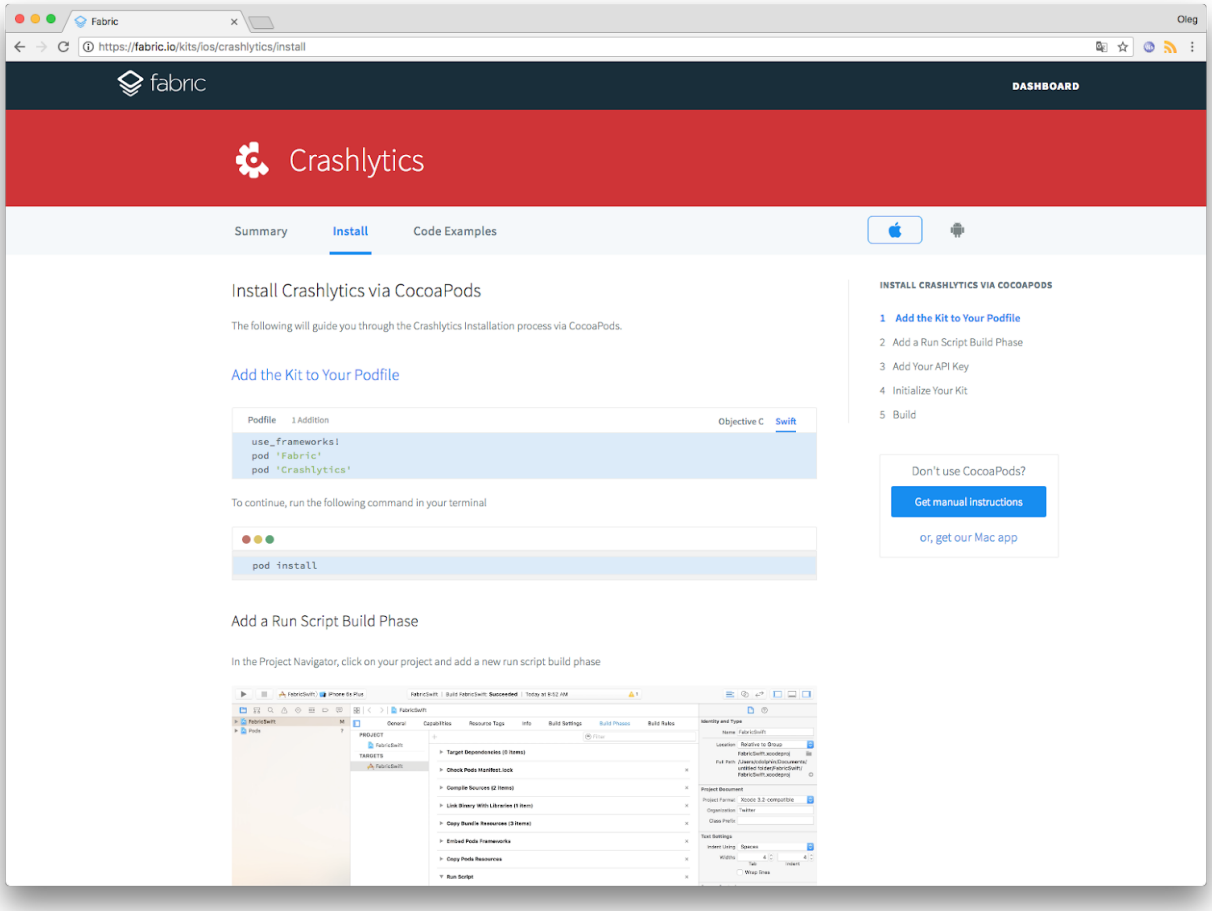
Crashlytics — один из инструментов сервиса Fabric. Это мощное, но легкое решение для фиксации сбоев в работе приложения.

К концу 2016 года Fabric обслуживал более 2 миллиардов активных устройств (практически все в мире) и обрабатывал 310 миллиардов сеансов приложений в месяц. Благодаря Crashlytics и его службе мобильной аналитики Fabric занял первое место по MightySignal как наиболее эффективное решение для отчетов о сбоях и мобильных аналитических решений среди 200 лучших приложений iOS.

Crashlytics дает мощную отчетность о сбоях, позволяет включить аналитику в реальном времени, которая поможет понять, что происходит в приложении.

Добавим Crashlytics к нашему проекту. Для этого зарегистрируемся на сайте fabric.io. Будем следовать инструкции установки, указанной в fabric.io/kits/ios/answers/install.


Fabric
Oleg
Защищено | https://fabric.io/kits?utm_campaign=discover&utm_medium=natural
Jump right to the kit you need
Crashlytics
The most powerful, yet lightest weight crash reporting solution.
Answers
The analytics engine of Fabric.
AdMob by Google
Mobile app monetization the smart way.
Amazon Cognito Sync
Develop apps quickly. Scale and run reliably.
Appsee
Analyze user behavior with videos of sessions, heatmaps & analytics.
Branch
Deep links to drive app installs, user engagement and retention across all channels.
GameAnalytics
To build great games, you need to understand player behavior.
Mapbox
Build experiences for exploring the world
Optimizely
Mobile A/B Testing
PubNub
Realtime apps made simple.
Stripe
Seamlessly integrated mobile payments.
Zendesk
Offer in-app self-service and support



Подключим **pods Crashlytics** в **Podfile**.

```
pod 'Fabric'
pod 'Crashlytics'
```

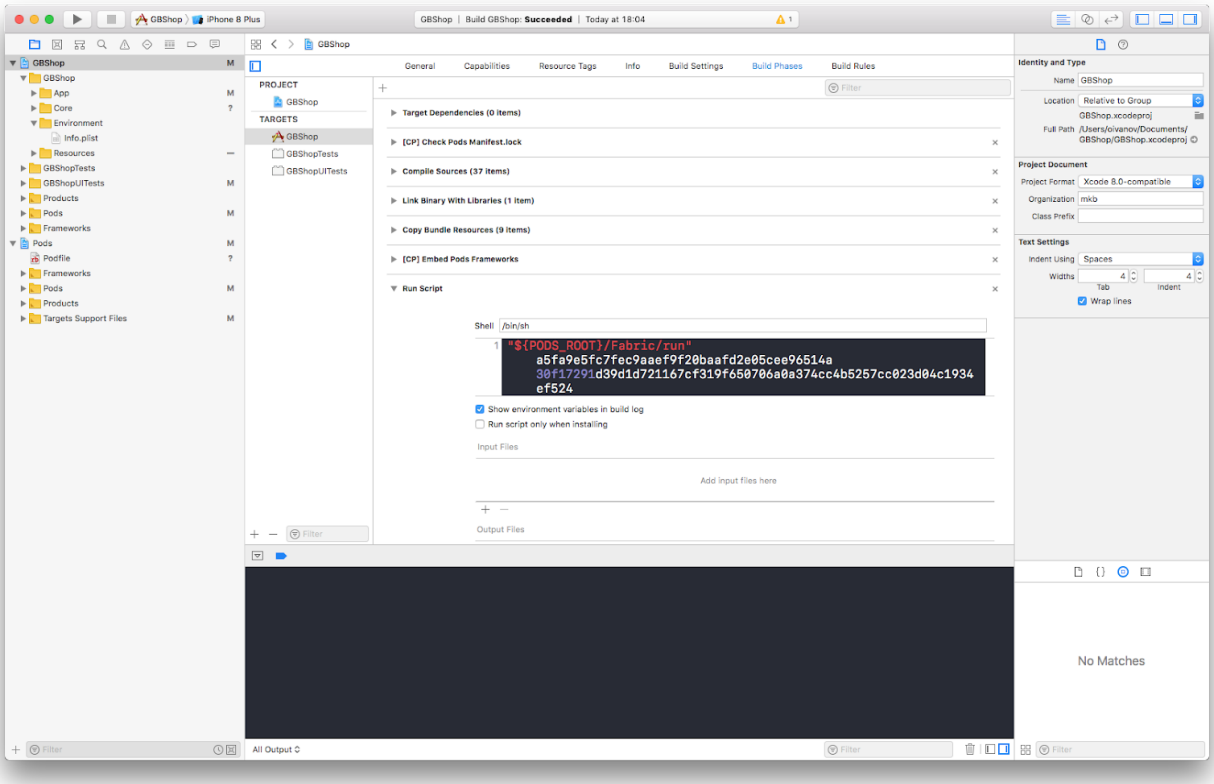
Выполним в терминале команду:

```
pod update
```

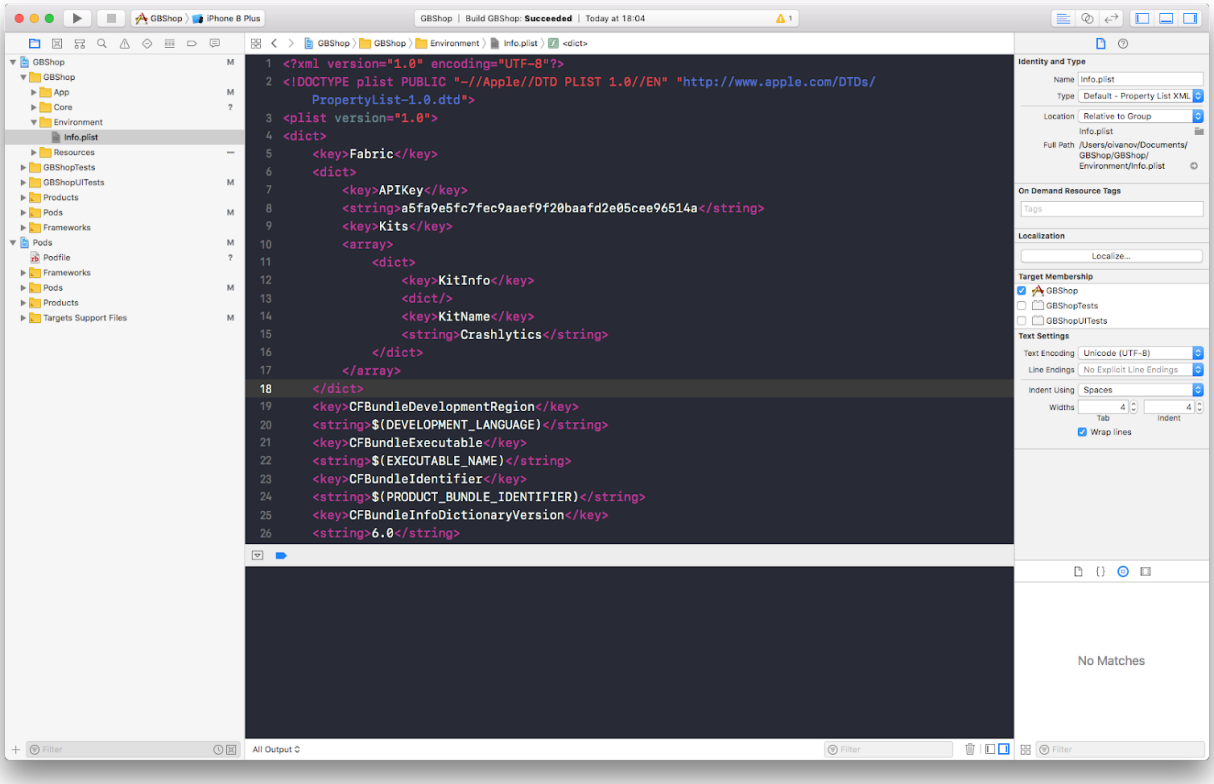
Далее добавим скрипт сценария сборки:

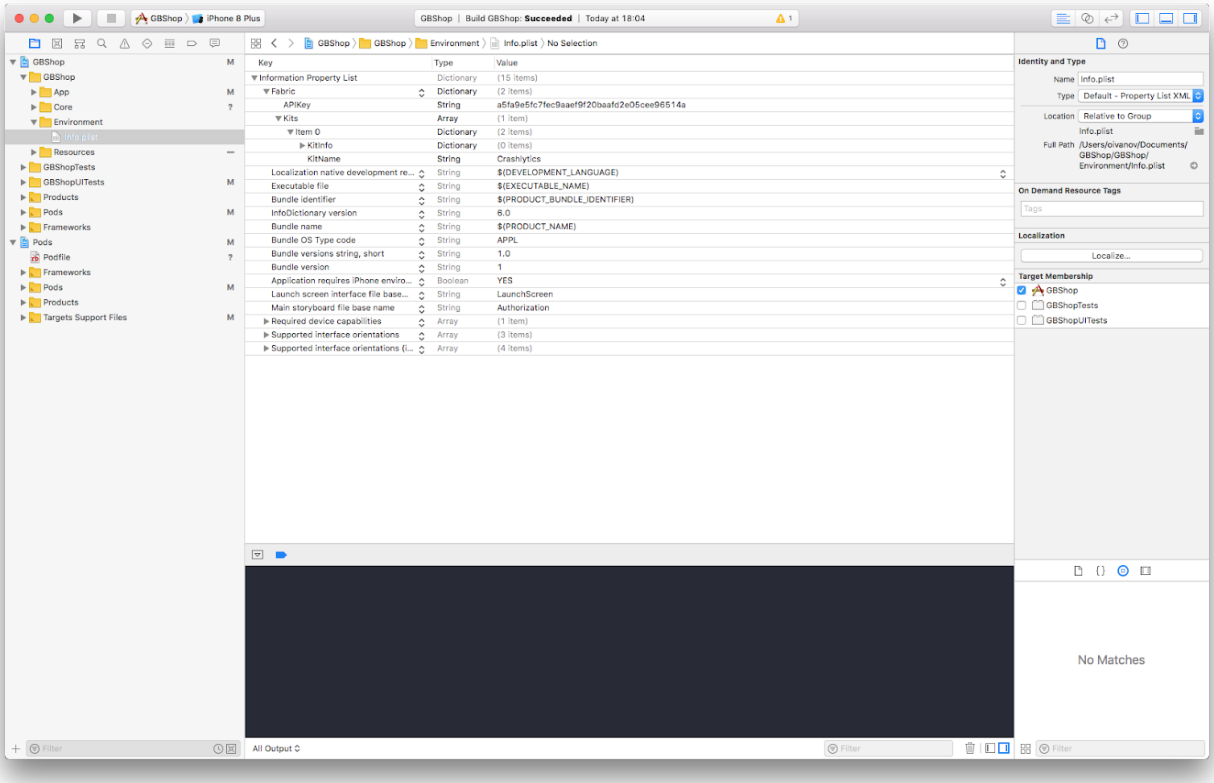
```
"${PODS_ROOT}/Fabric/run" <FABRIC_API_KEY> <BUILD_SECRET>
```

Параметры **<FABRIC_API_KEY> <BUILD_SECRET>** будут доступны после входа, и их значения будут вставлены в инструкцию установки на странице fabric.io/kits/ios/answers/install.



Следуем инструкции и добавляем **API Key** в **info.plist** проекта.





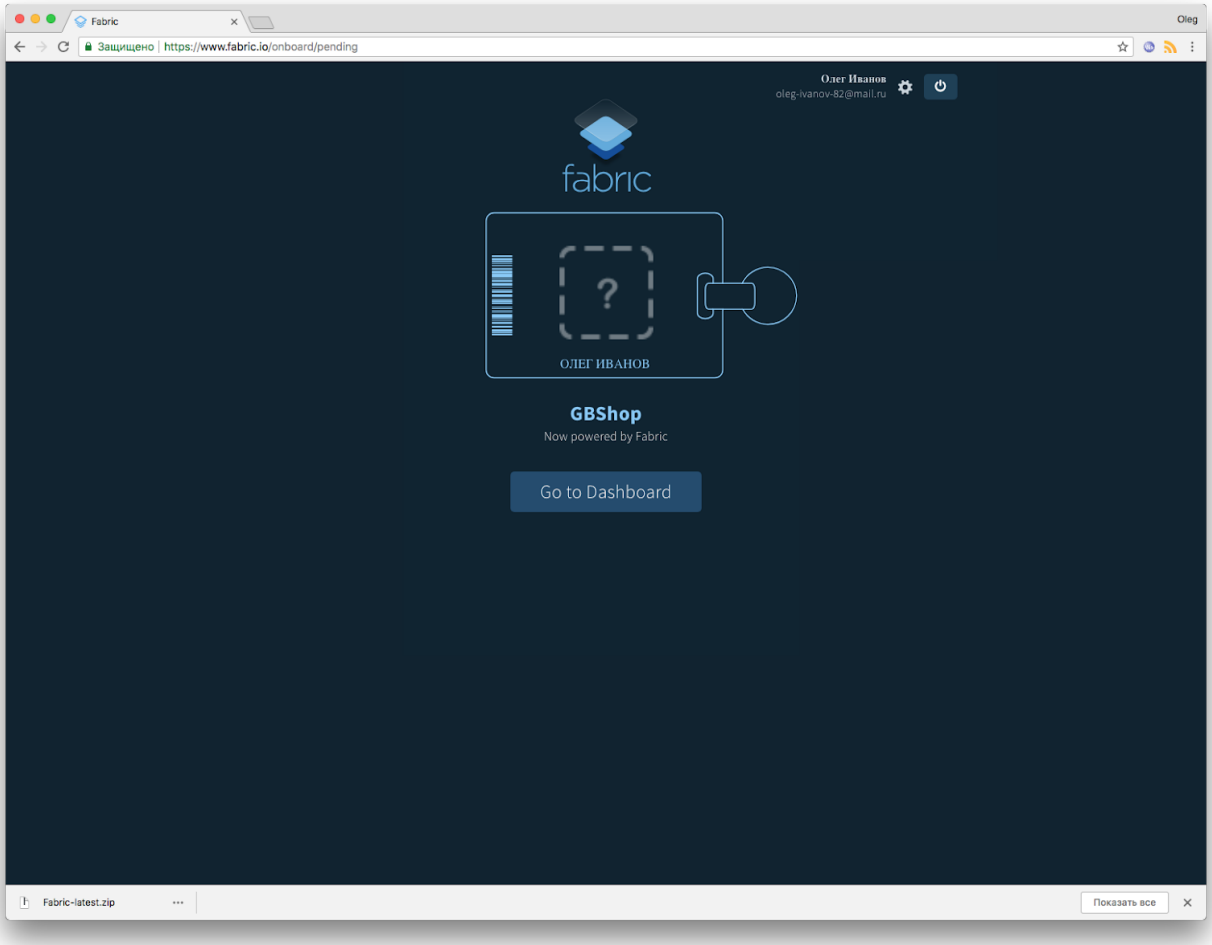
Проводим инициализацию **Fabric** в **AppDelegate**.

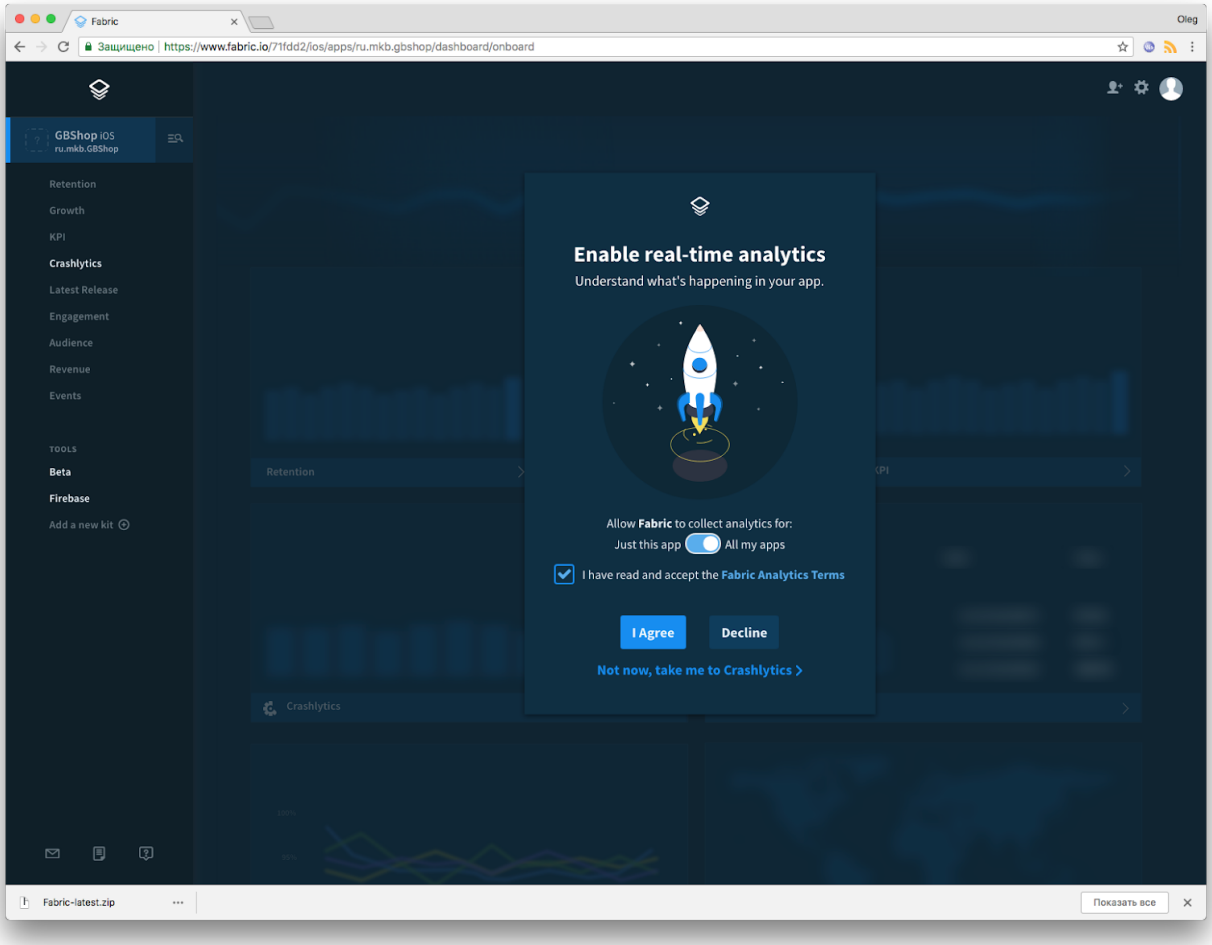
```
import UIKit
import Alamofire
import Fabric
import Crashlytics

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        Fabric.with([Crashlytics.self])

        return true
    }

...

}
```

Теперь Crashlytics готов к использованию в проекте.

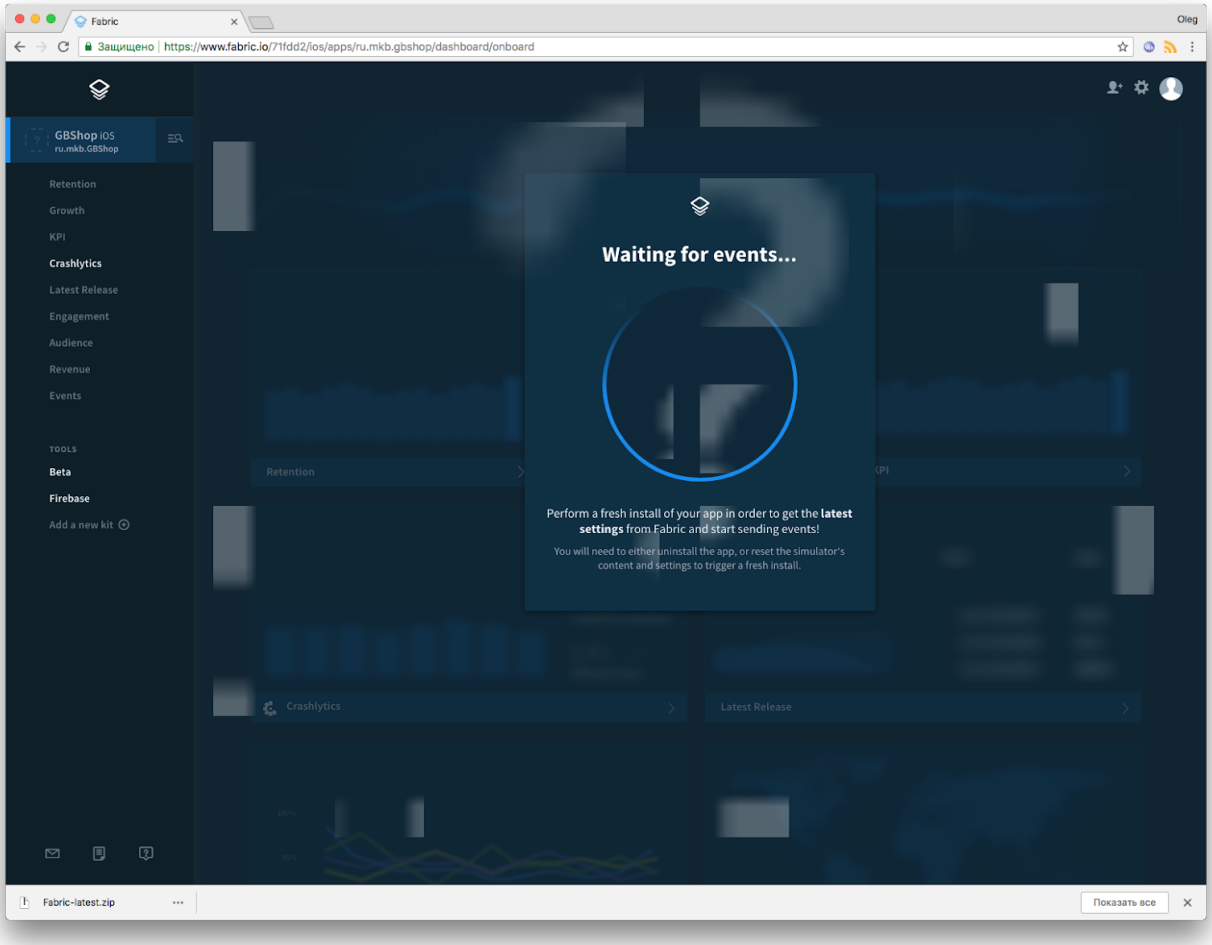
Подключение проекта к Fabric

После входа на сайте fabric.io предложат подключение проекта. После описанных выше действий и запуска приложения будет доступен **dashboard**.


Fabric
Oleg
Защищено | https://www.fabric.io/onboard/pending
Олег Иванов
oleg-ivanov-82@mail.ru
fabric
?
ОЛЕГ ИВАНОВ
GBShop
Now powered by Fabric
Go to Dashboard
Fabric-latest.zip
Показать все



Далее Fabric будет ожидать «события» от приложения для старта работы с ним.



Создадим данное событие в проекте с помощью **Answers**.

Answers

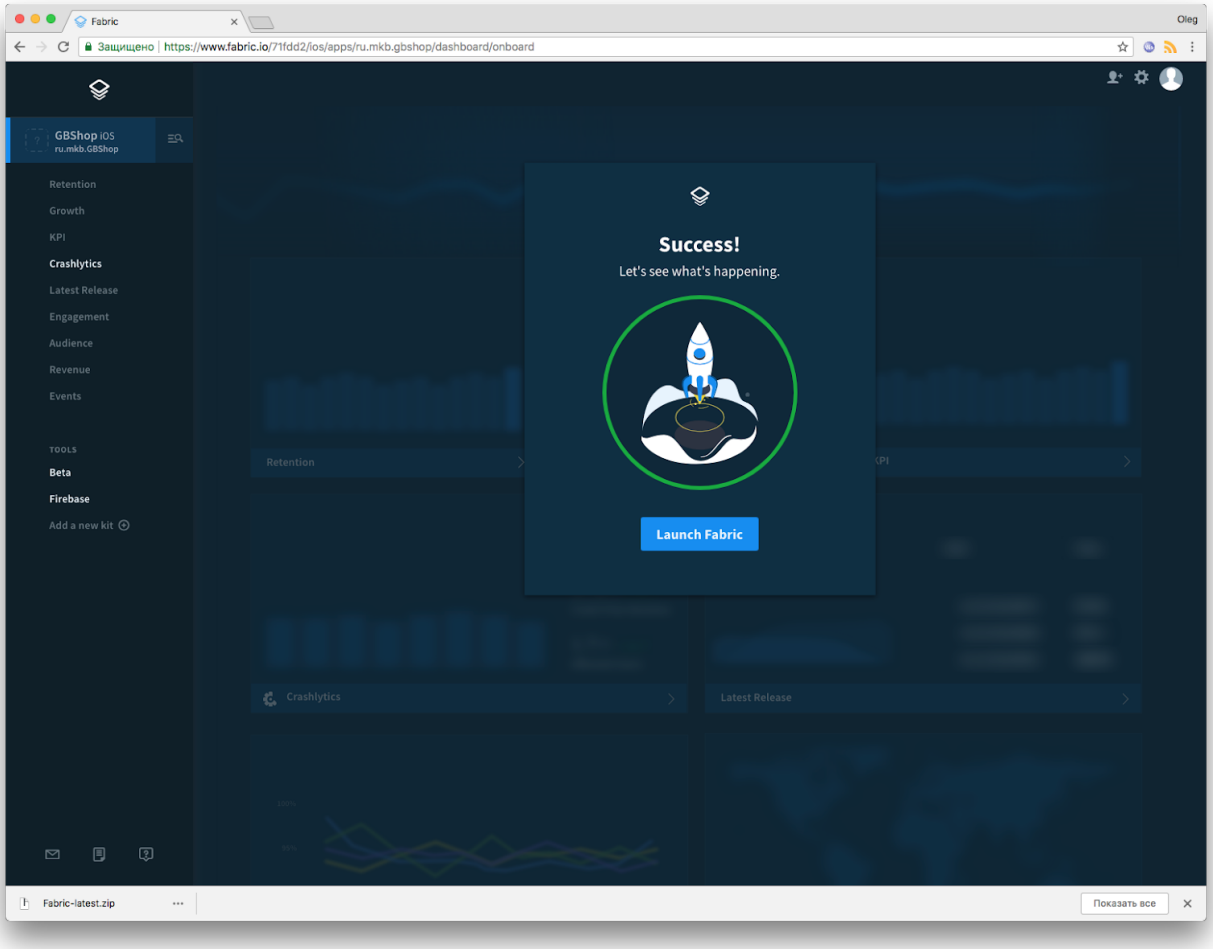
Воспользуемся инструментом Answers — движком мобильной аналитики от Fabric. Answers — это SDK.

Добавим первое стандартное событие: регистрацию входа в систему — **logLogin()**. Для этого вставим представленный ниже код в **AppDelegate**.

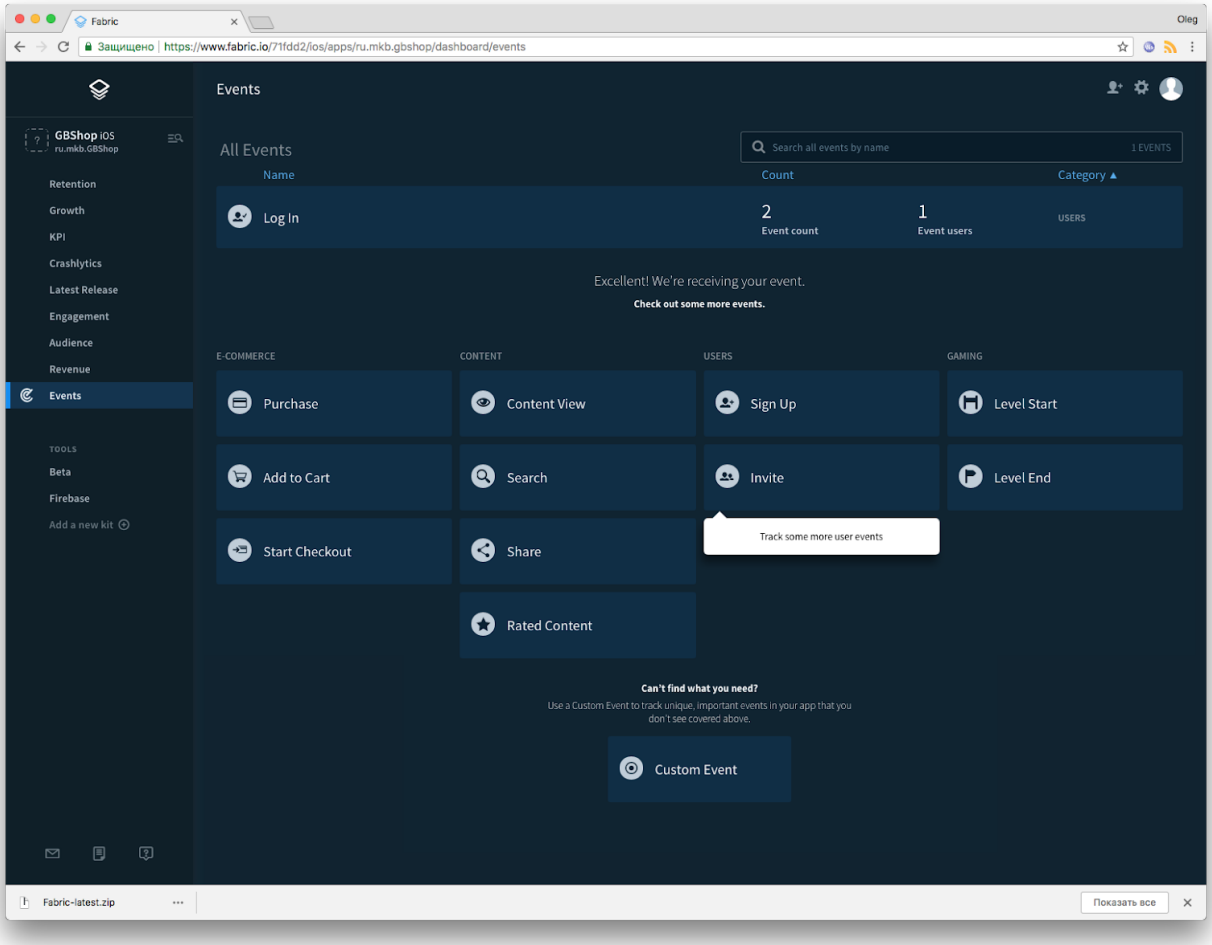
```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        Fabric.with([Crashlytics.self])

        Answers.logLogin(withMethod: "default", success: true,
customAttributes: nil)

        return true
    }
```

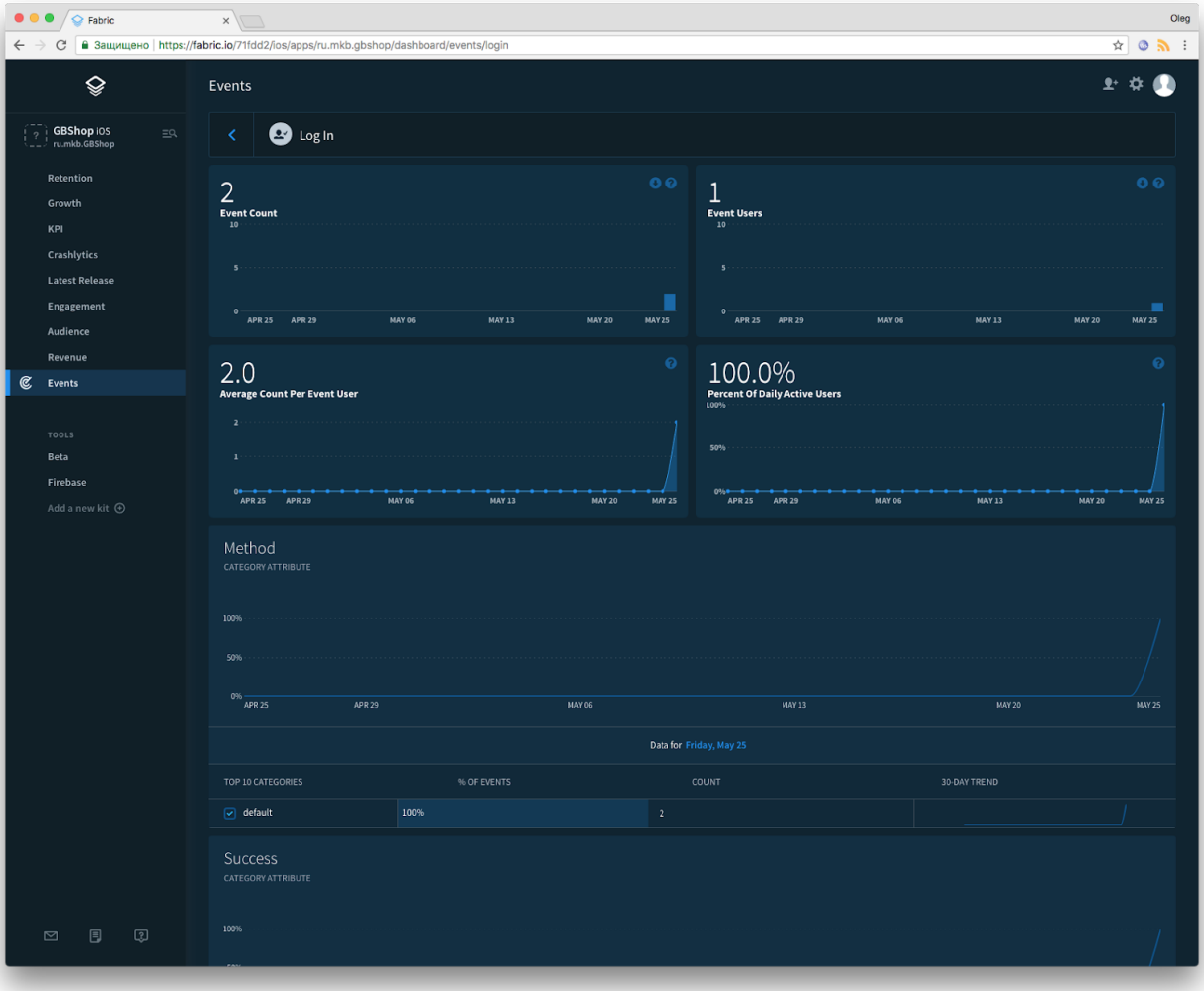
```
...

}
```

Запускаем приложение и видим в **dashboard** успешный прием события **logLogin()**. Обратите внимание — событие может не сразу «дойти» до ресурса Fabric.



Перейдя к вкладке **Events**, увидим событие и графики статистики его появления в проекте.


Fabric
Oleg
Защищено | https://www.fabric.io/71fdd2/ios/apps/ru.mkb.gbshop/dashboard/events
Events
GBShop iOS
ru.mkb.GBShop
Retention
Growth
KPI
Crashlytics
Latest Release
Engagement
Audience
Revenue
Events
TOOLS
Beta
Firebase
Add a new kit
All Events
Search all events by name
1 EVENTS
Name
Count
Category
Log In
2
Event count
1
Event users
USERS
Excellent! We're receiving your event.
Check out some more events.
E-COMMERCE
CONTENT
USERS
GAMING
Purchase
Content View
Sign Up
Level Start
Add to Cart
Search
Invite
Level End
Start Checkout
Share
Track some more user events
Rated Content
Can't find what you need?
Use a Custom Event to track unique, important events in your app that you don't see covered above.
Custom Event
Fabric-latest.zip
Показать все

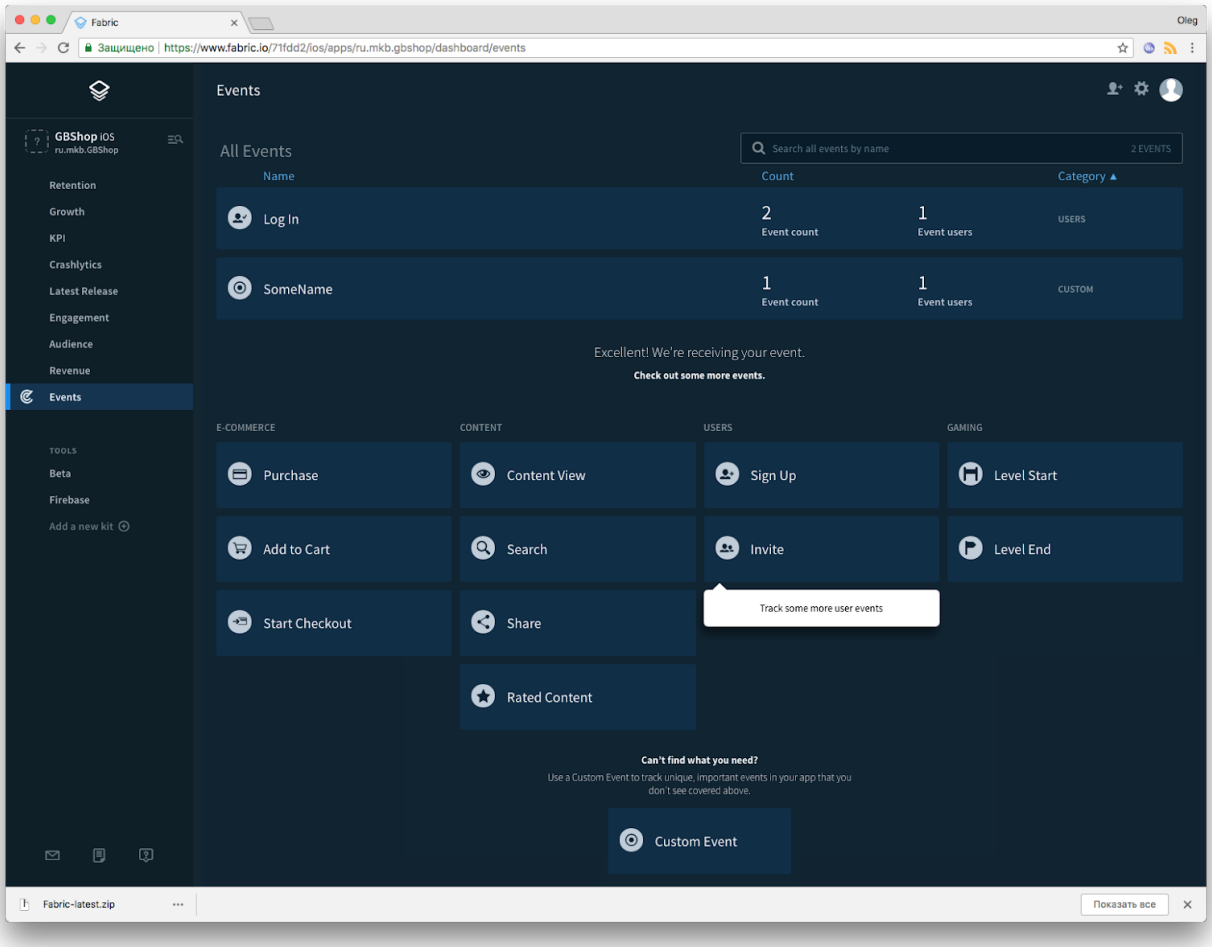


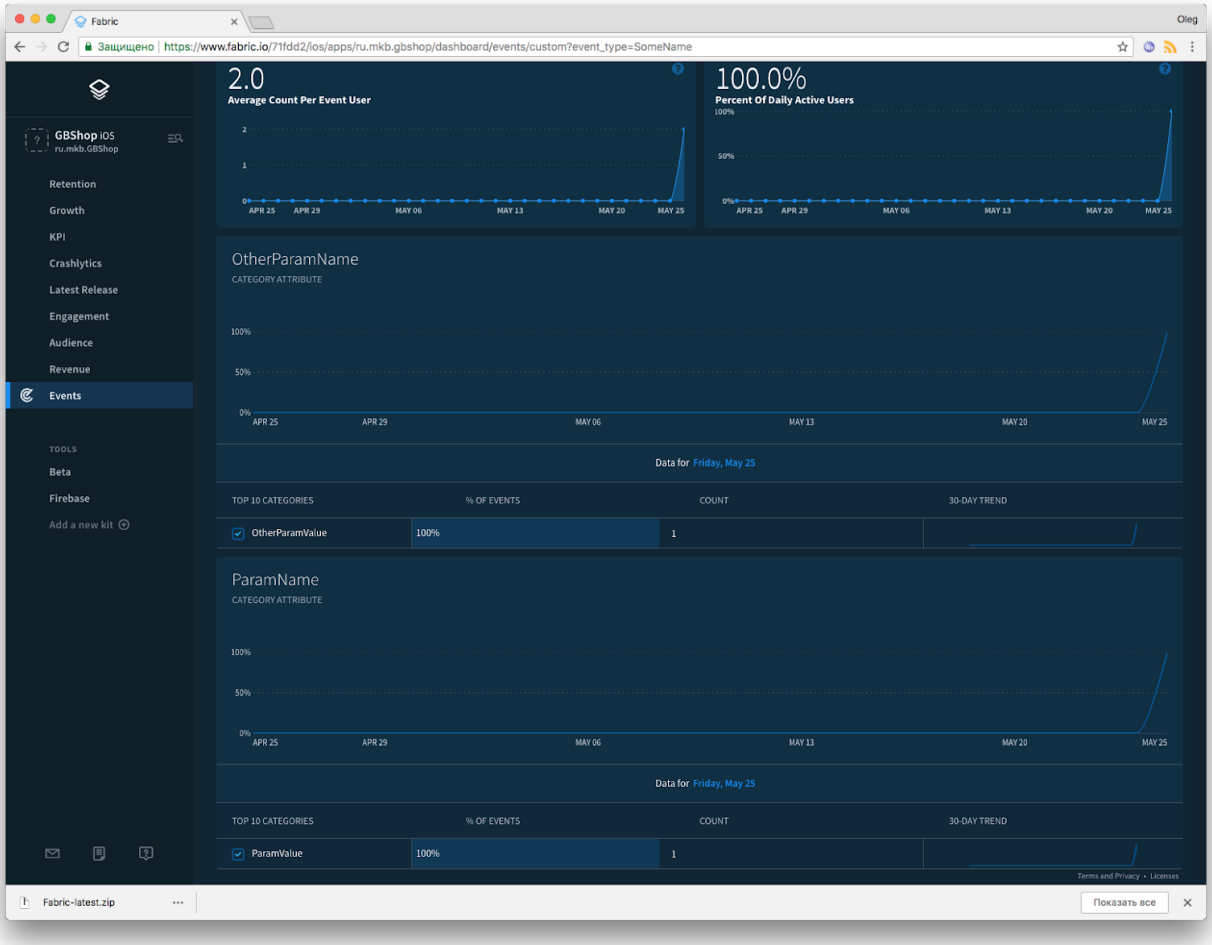
Добавим еще одно событие, но не стандартное, а настраиваемое, с двумя параметрами.

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        Fabric.with([Crashlytics.self])

        Answers.logLogin(withMethod: "default", success: true, customAttributes:
nil)

        Answers.logCustomEvent(
            withName: "SomeName",
            customAttributes: ["ParamName" : "ParamValue", "OtherParamName" :
"OtherParamValue"]
        )

        return true
    }

...
```

```
}
```

Выполним запуск приложения и перейдем на **dashboard**:





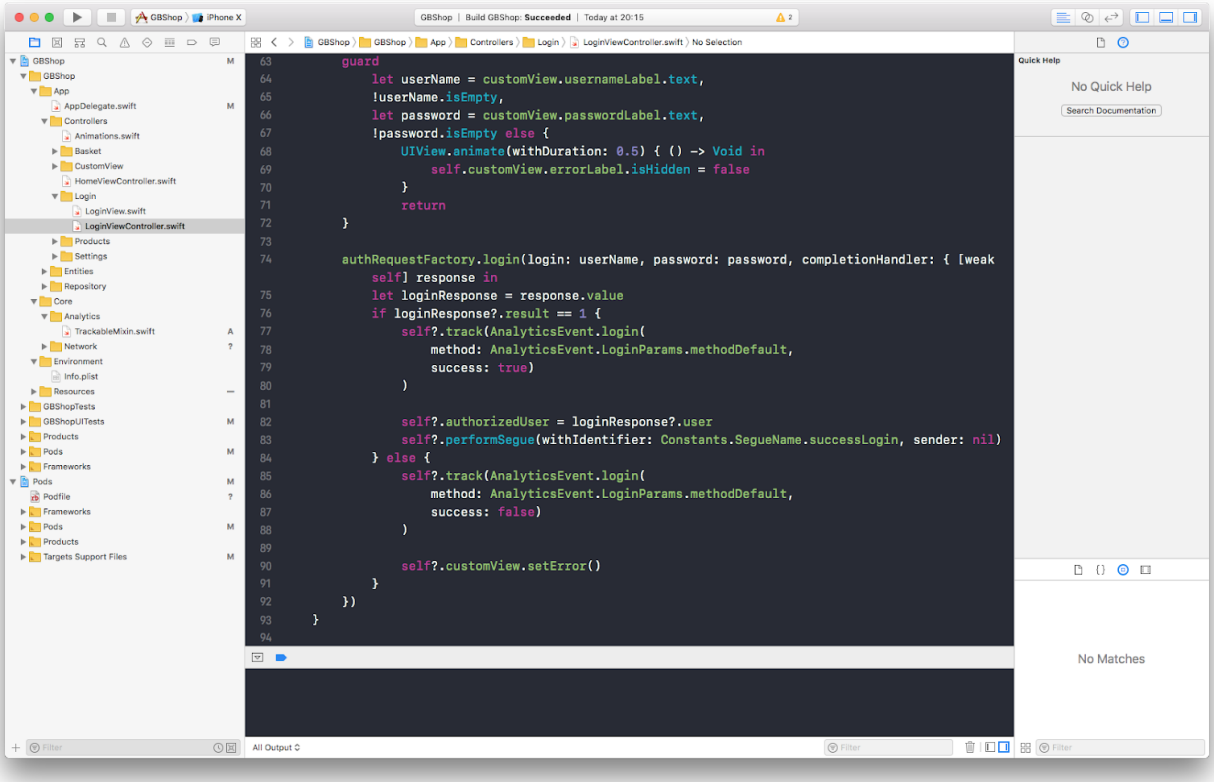
Используя Answers, можем добавить события на нужные моменты в проекте. Но напрямую использовать Answers неверно. А если придется перейти на другую систему сбора статистики или использовать одновременно несколько? Выделим отдельный класс, который будет предоставлять api для формирования событий и скрывать в себе реализацию.

```
import Foundation
import Crashlytics

enum AnalyticsEvent {
    enum LoginParams {
        static let methodDefault = "default"
    }

    enum SomeMethodParams {
        static let nameDefault = "default"
        static let nameAssertionFailure = "assertionFailure"
    }

    case login(method: String, success: Bool)
    case someMethod(name: String, some: String)
}

protocol TrackableMixin {
    func track(_ event: AnalyticsEvent)
}

extension TrackableMixin {
```

```
    func track(_ event: AnalyticsEvent) {
        switch event {
        case let .login(method, success):
            let success = NSNumber(value: success)
            Answers.logLogin(withMethod: method, success: success,
customAttributes: nil)
        case let .someMethod(name, some):
            Answers.logCustomEvent(withName: name, customAttributes:
["parameter" : some])
        }
    }
}
```

Добавим «настоящее» применение события **logLogin()** в контроллер аутентификации.

```
…

authRequestFactory.login(login: userName, password: password, completionHandler:
{ [weak self] response in
    let loginResponse = response.value
    if loginResponse?.result == 1 {
        self?.track(AnalyticsEvent.login(
            method: AnalyticsEvent.LoginParams.methodDefault,
            success: true)
        )

        self?.authorizedUser = loginResponse?.user
        self?.performSegue(withIdentifier: Constants.SegueName.successLogin,
sender: nil)
    } else {
        self?.track(AnalyticsEvent.login(
            method: AnalyticsEvent.LoginParams.methodDefault,
            success: false)
        )

        self?.customView.setError()
    }
})

…
```



Еще один интересный пример применения Answers — использование **assertionFailure**. Данная функция предназначена для проверки работоспособности приложения. Ее вызов приводит к его остановке, но только при сборке в Debug-конфигурации. Если хотим иметь подобный инструмент отслеживания, но без остановки приложения и в Release-конфигурации, на помощь приходит Answers.

Допустим, у нас был такой код с использованием **assertionFailure**:

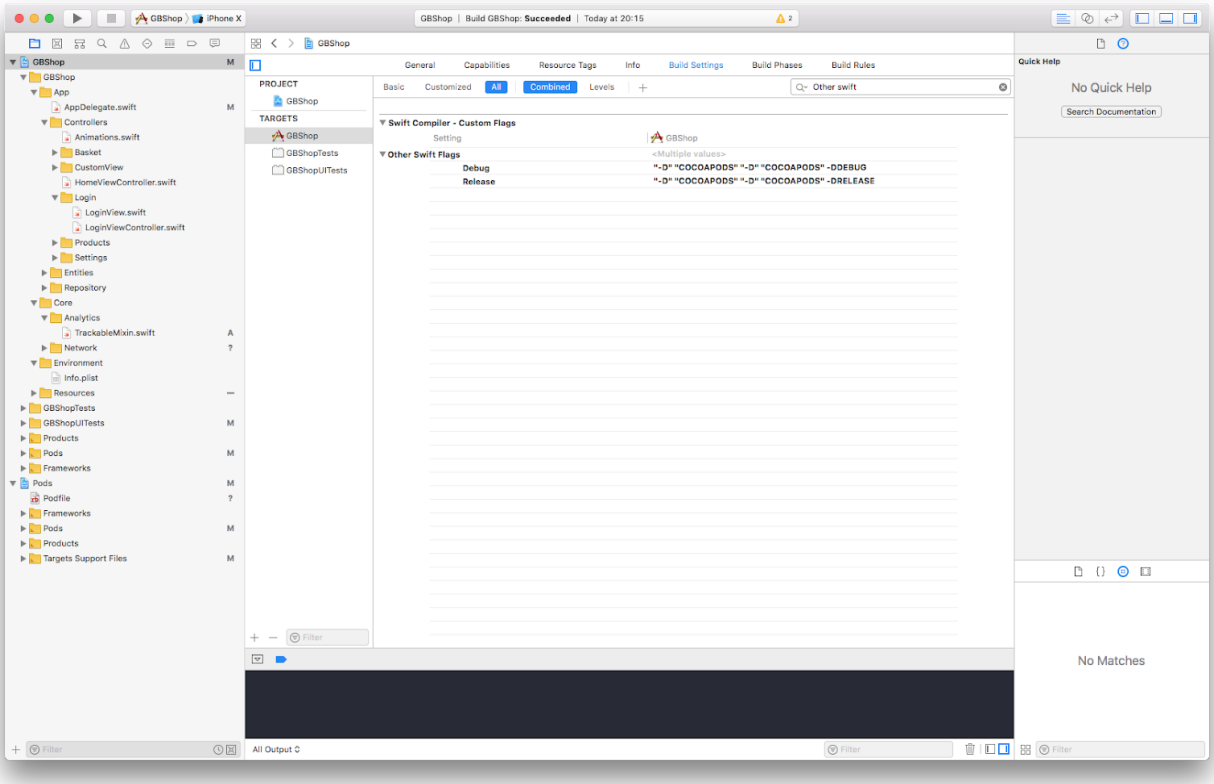
```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == Constants.SegueName.successLogin {
        guard
            let tabBarController = segue.destination as? UITabBarController else
{

                assertionFailure("Some warning text")

                return
        }
    }

...

}
```

Написав свой метод **assertionFailure** и создав событие Answers, сможем отслеживать подобные моменты и оперативно их исправлять:

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == Constants.SegueName.successLogin {
        guard
            let tabBarController = segue.destination as? UITabBarController else
{
```

```
                assertionFailure("Some warning text")

                return
            }
        }

...

}

func assertionFailure(_ message: String) {
    #if DEBUG
        Swift.assertionFailure(message)
    #else
        track(AnalyticsEvent.someMethod(
            Answers.logCustomEvent(
                withName: "AssertionFailure",
                customAttributes: ["message" : message]
            )
        )
    #endif
}
```

Для данной реализации необходимо задать в настройках сборки проекта **Other Swift Flags**. В этом разделе зададим макросы препроцессорной обработки **DEBUG** и **RELEASE** для соответствующих конфигураций. Обратите внимание — перед именем макроса используем префикс «**-D**».



Практическое задание

1. Использовать **assertionFailure** в приложении.
2. Добавить аналитику на основные действия пользователя:
    a. Успешный вход;
    b. Неуспешный вход;
    c. Выход;
    d. Регистрация;
    e. Открытие списка товаров;
    f. Открытие конкретного товара;
    g. Добавление товара в корзину;
    h. Удаление товара из корзины;
    i. Оплату заказа;
    j. Добавление отзыва.

Дополнительные материалы

1. fabric.io.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Стив Макконнелл. Совершенный код.