StoreKit

Seminar Games Development (WS13)

04.10.2013

Session 15

"Tell me and I will forget.
Show me and I will remember.
Involve me and I will understand.
Step back and I will act."

Agenda

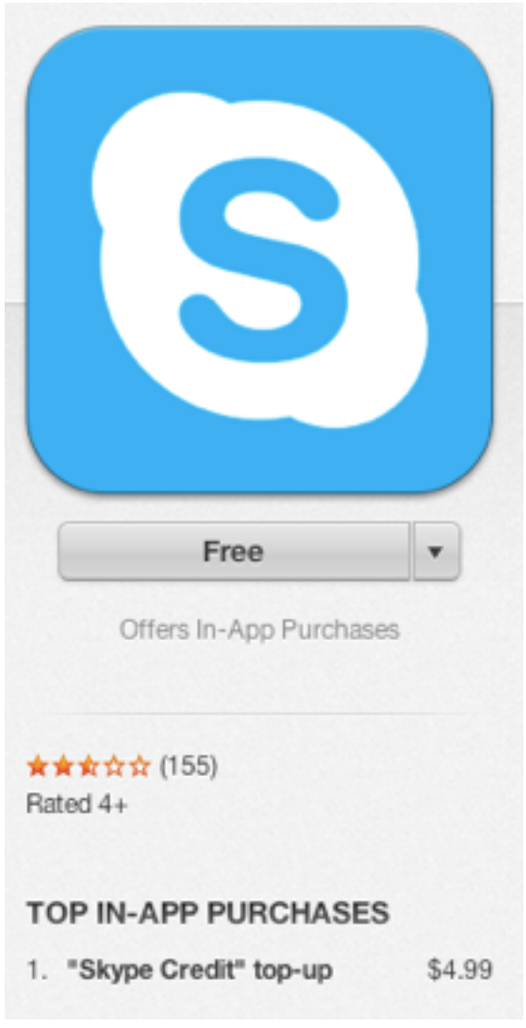
- Motivation / Examples (in) applications
- Overview
  - What is an In-App Purchase?
  - Overview: types of In-App Purchases
- Implementation
  - iTunes Connect Metadata Setup
  - StoreKit on iOS (+ exercise)
  - Testing in Sandbox
- Summary

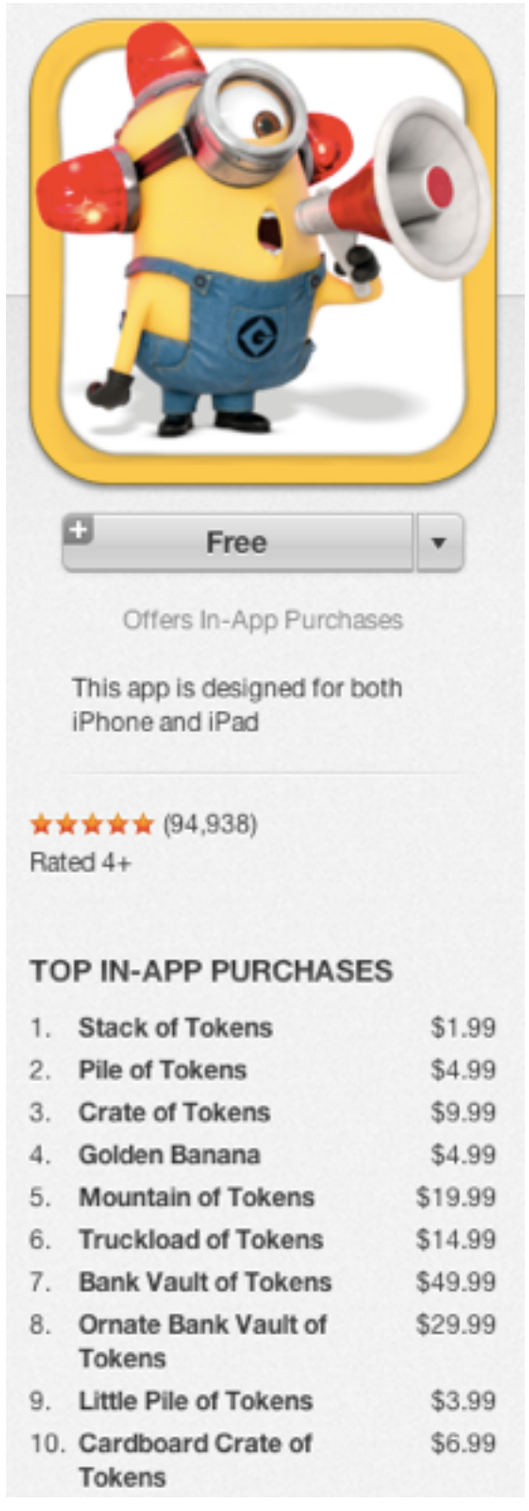
Agenda

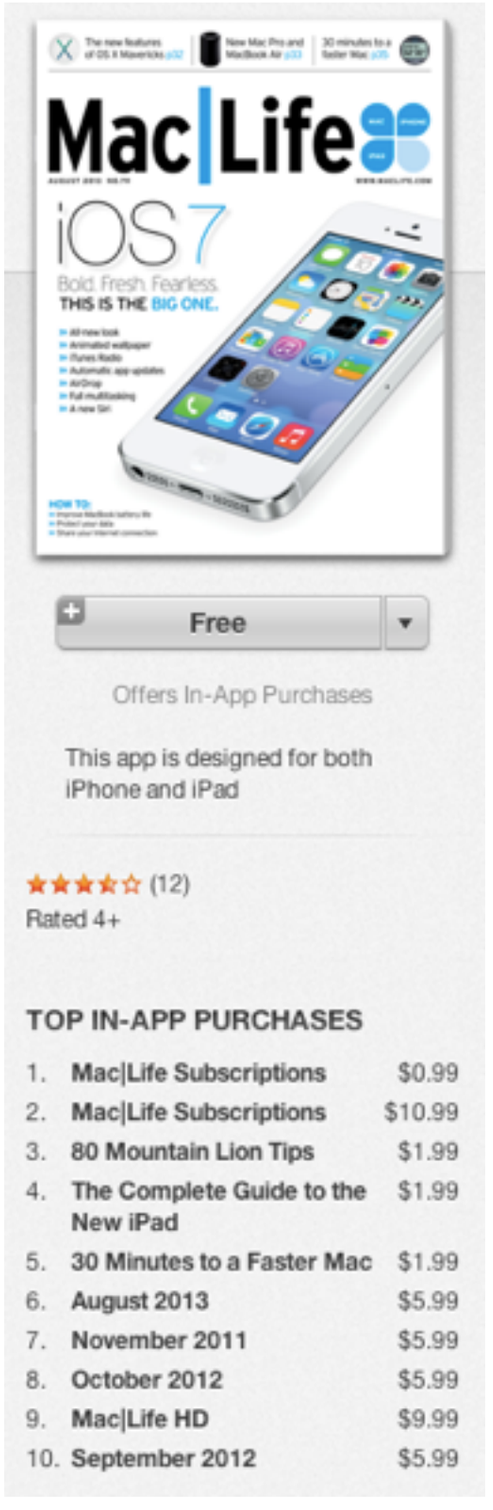
- **Motivation / Examples (in) applications**
- Overview
  - What is an In-App Purchase?
  - Overview: types of In-App Purchases
- Implementation
  - iTunes Connect Metadata Setup
  - StoreKit on iOS (+ exercise)
  - Testing in Sandbox
- Summary

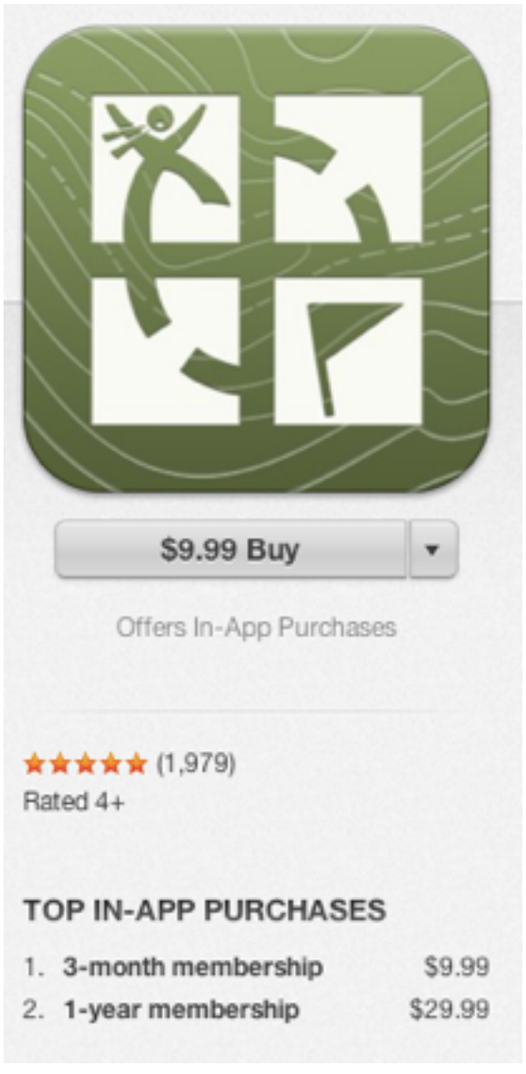
Examples applications

Examples applications

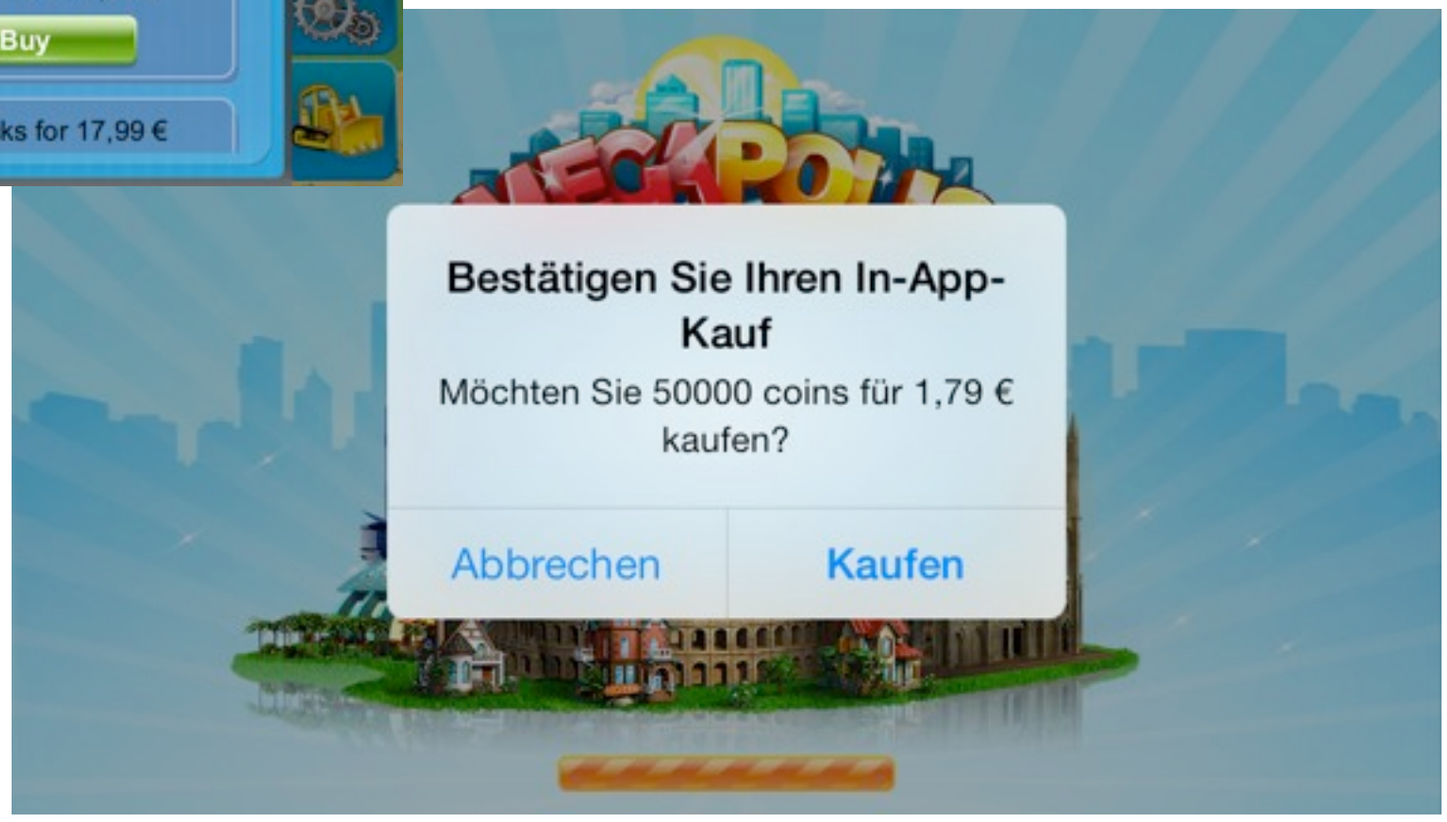




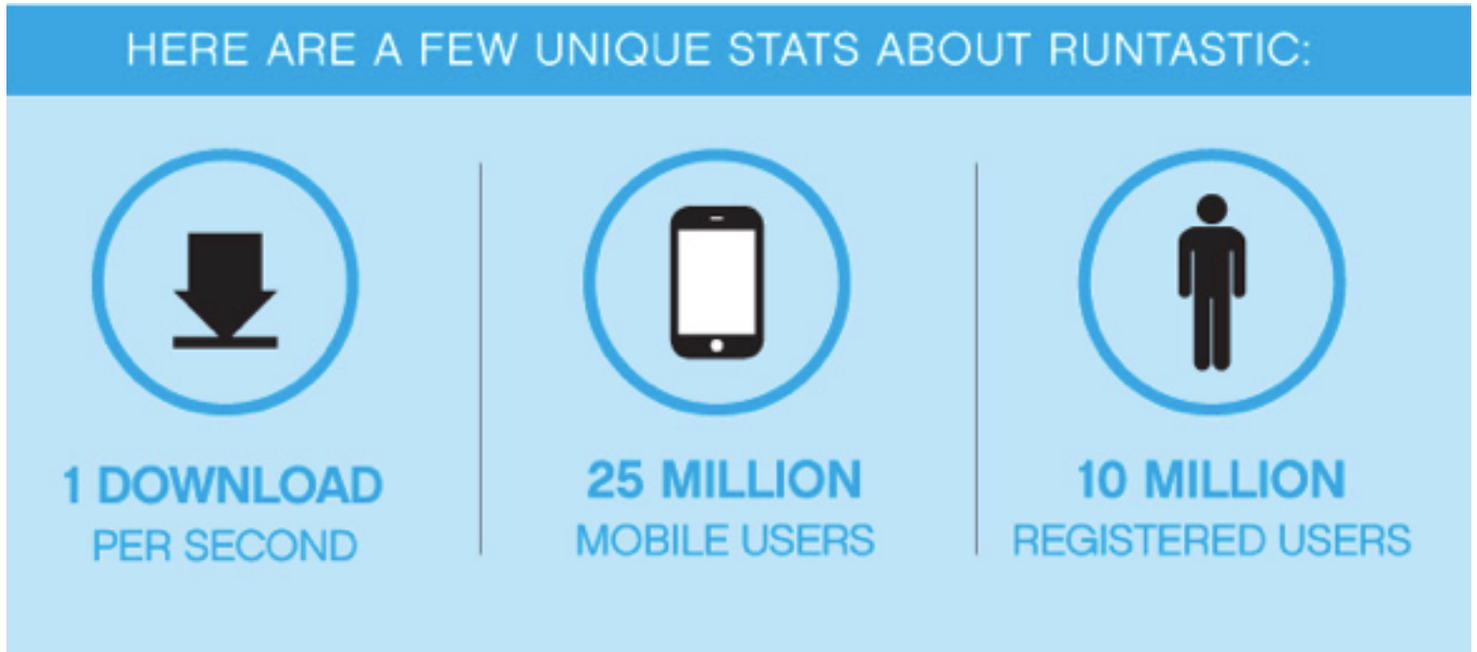


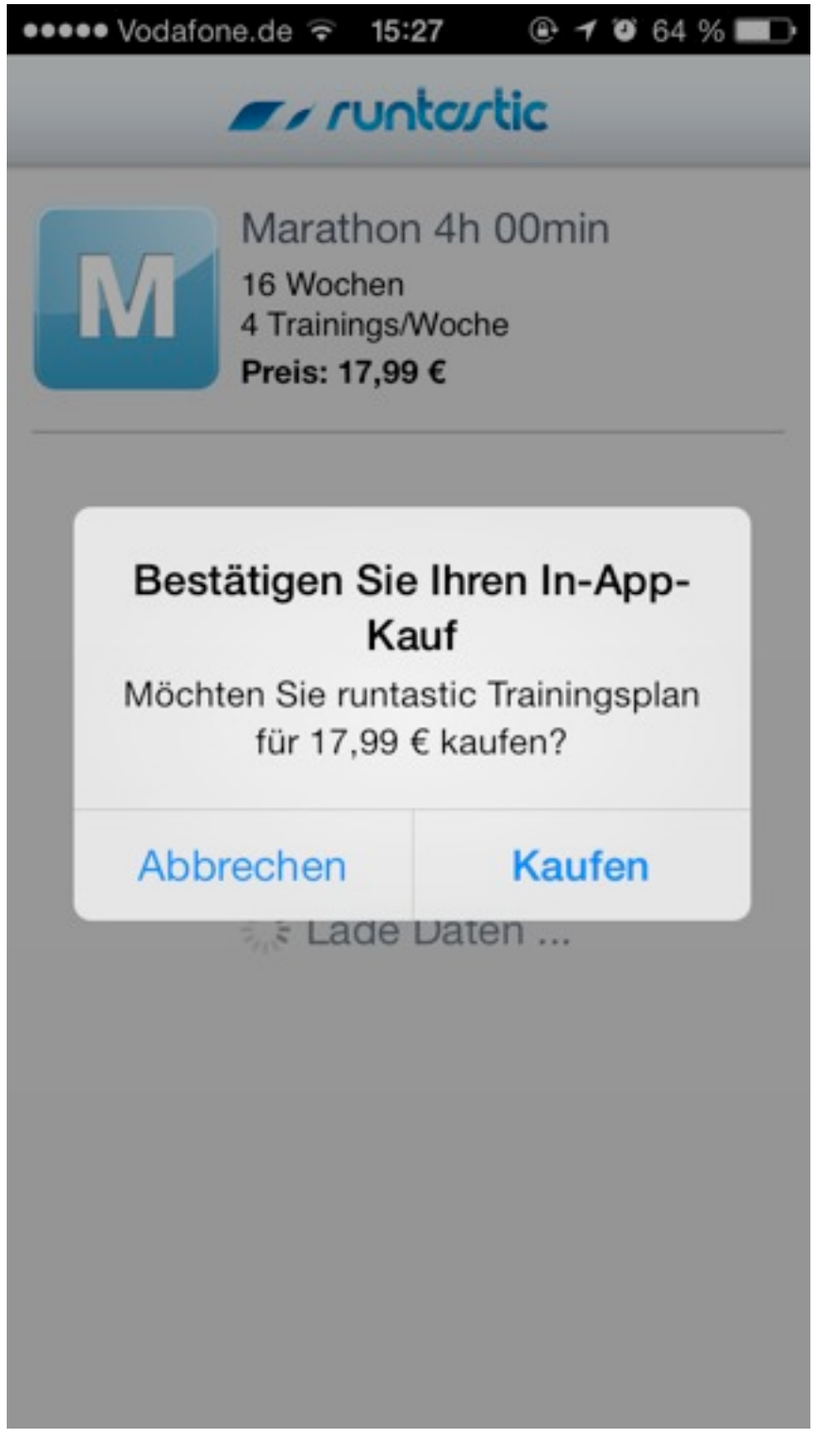


Examples in applications



Examples in applications





Motivation

- **Money!**
  - You can earn more money than just one time the price of your app!
  - Some users are willing to spend a lot more on extra content („freemium“)!
- **Market Share / New Business Models!**
  - You can release your app for free, which is a no-brainer download for most people → more downloads / users reached!
  - If users enjoy your app, they can purchase more in-app functionality!
- **Easy Maintenance!**
  - You can keep adding additional content in the future to the same app, rather than having to make a new app to earn more money!

Motivation

„It allows us to reach many people who otherwise have a fear to purchase games.“

Jens Begemann, Co-Founder and CEO of
Wooga, the biggest European Social Games Developer

Source: Handelsblatt, August 22, 2013, Nr. 161 and wooga.com

Agenda

- Motivation / Examples (in) applications
- **Overview**
  - **What is an In-App Purchase?**
  - Overview: types of In-App Purchases
- Implementation
  - iTunes Connect Metadata Setup
  - StoreKit on iOS (+ exercise)
  - Testing in Sandbox
- Summary

What is an In-App Purchase?

- In-App Purchase lets you **sell a variety of items** directly within your free or paid app:
  - adding for pay features to a free app
  - adding additional levels to a game
  - purchasing virtual goods
  - adding magazine subscriptions
  - etc.

- Implemented using the StoreKit API, introduced with iOS 3.0 and Mac OS 10.7 (In-App Purchase works for Mac Apps too)

Dos and Don’ts

Do

- deliver your **digital good or service** within your app.
- as for all apps: give users an added value.

Don‘t

- use In-App Purchase to sell **real-world goods and services**.
- offer items for purchase that contain, or relate to, pornography, hate speech, defamation ... same guidelines as for app admission!
- In-App Purchase items cannot be shared across applications or platforms.

Agenda

- Motivation / Examples (in) applications
- **Overview**
  - What is an In-App Purchase?
  - **Overview: types of In-App Purchases**
- Implementation
  - iTunes Connect Metadata Setup
  - StoreKit on iOS (+ exercise)
  - Testing in Sandbox
- Summary

Overview: types of In-App Purchases

- Non-consumable
- Consumable
- Non-renewing subscription
- Auto-renewing subscription

Non-consumable

- Products designed to be **purchased once**, comparable to a paid app paid once.
- Consider these purchases as durable, persistent.
- Can be **restored** on all user‘s devices… managed by StoreKit.
- Developer is responsible for ensuring purchase is present on all devices using `restoreCompletedTransactions` (we‘ll get to that later).
- For example:
  - new level in a game
  - additional feature in an app
  - ad-freeness

Consumable

- Products designed to be **purchased multiple times** by the user.
- **No restore** possible.
- For example:
  - items consumed as part of game play, i.e. power-ups, coins, etc.
  - virtual currency as a means of advancement (check AppStore admission guidelines).

Non-renewing subscription

- Usually not implemented anymore, since Apple's documentation seems to suggest that they are deprecated in favor of auto-renewing subscriptions.

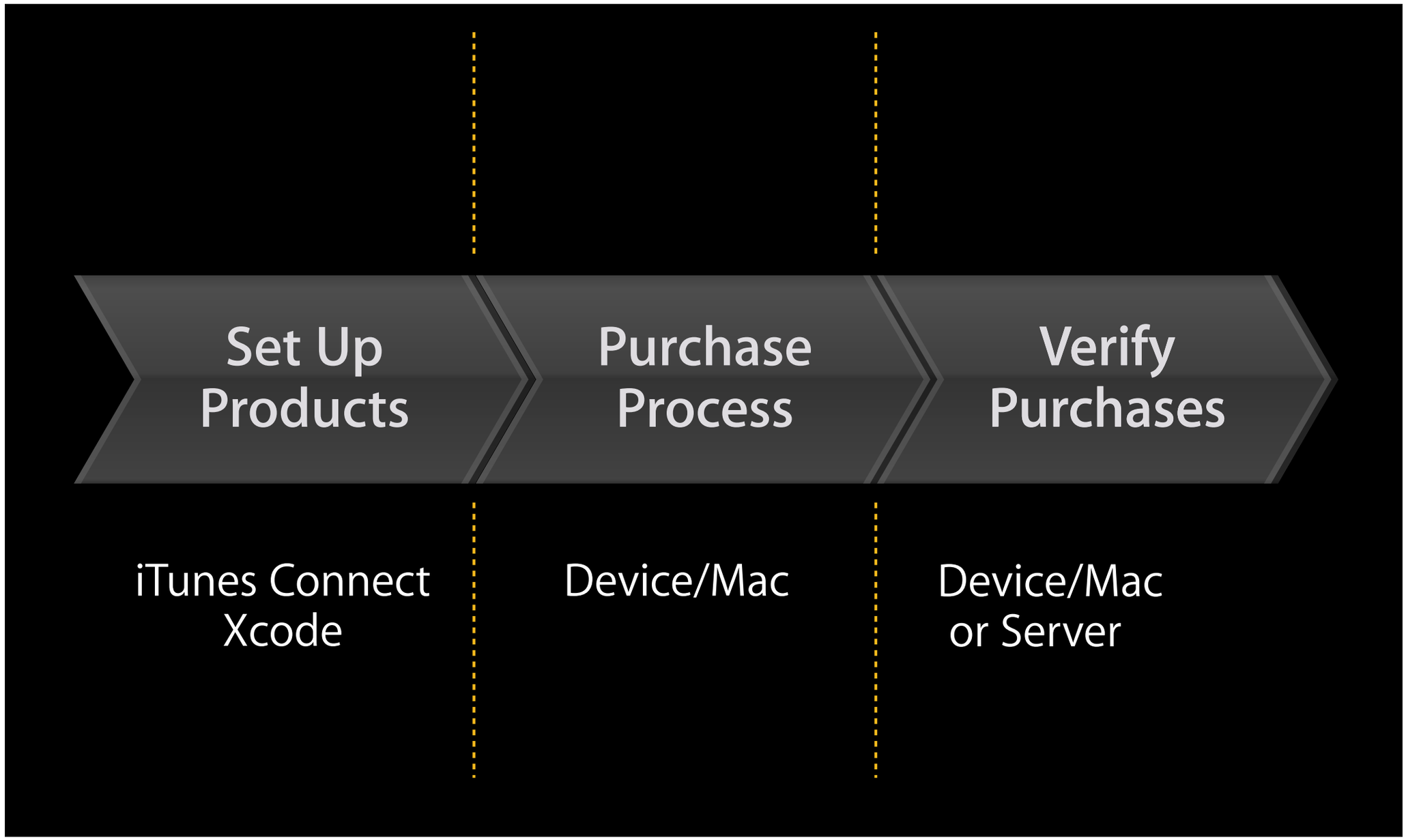
Auto-renewing subscription

- Enables **true subscription support** within your app.
- Subscription renewed at the end of the subscription period
  - Customer given option to opt out in iTunes.
- New content delivered to all devices with same customer Apple ID.
- Duration options:
  - 7 days, 1 month, 2 months, 3 months, 6 months, 1 year.
- Incentive for users to provide their email address (CRM ...)
- But: the most complex In-App purchase type to implement...

Agenda

- Motivation / Examples (in) applications
- Overview
  - What is an In-App Purchase?
  - Overview: types of In-App Purchases
- **Implementation**
  - iTunes Connect Metadata Setup
  - StoreKit on iOS (+ exercise)
  - Testing in Sandbox
- Summary

Implementing In-App Purchases

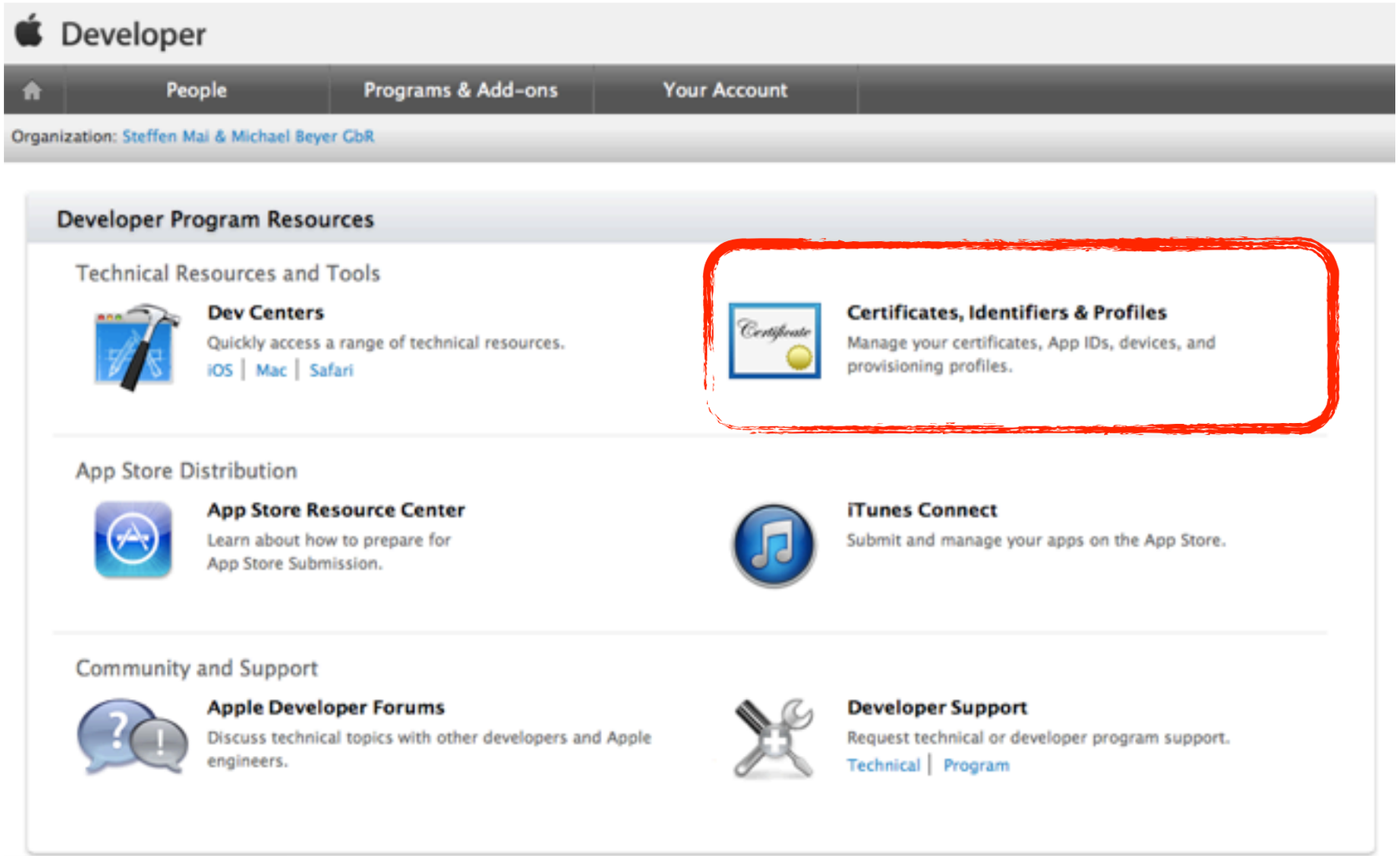


Agenda

- Motivation / Examples (in) applications
- Overview
  - What is an In-App Purchase?
  - Overview: types of In-App Purchases
- **Implementation**
  - **iTunes Connect Metadata Setup**
  - StoreKit on iOS (+ exercise)
  - Testing in Sandbox
- Summary

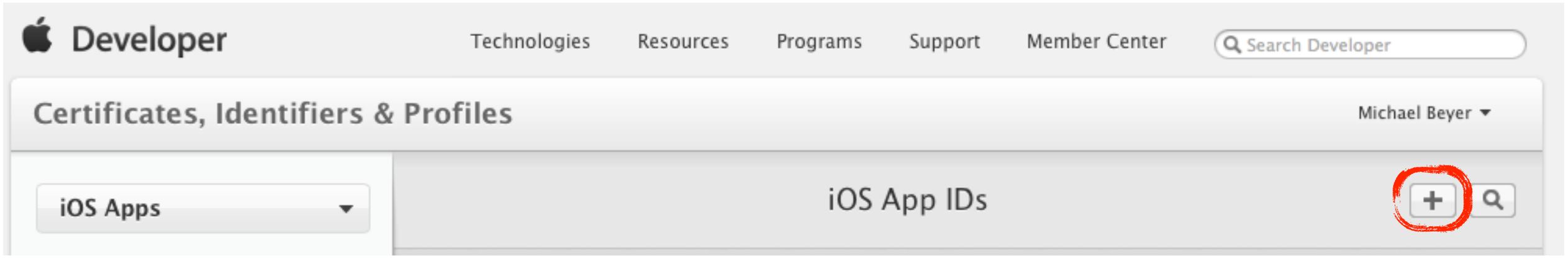
iTunes Connect (iTC) Metadata Setup

- Go to developer.apple.com and login to the Member Center
- Select „Certificates, Identifiers & Profiles“.



iTunes Connect (iTC) Metadata Setup

iTunes Connect (iTC) Metadata Setup



iTunes Connect (iTC) Metadata Setup

Each part of an App ID has different and important uses for your app. Learn More

App ID Description

Name: BuyFruit

You cannot use special characters such as @, &, *, ', "

App ID Prefix

Value: FR4[redacted] (Team ID)

App ID Suffix

Explicit App ID

If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.

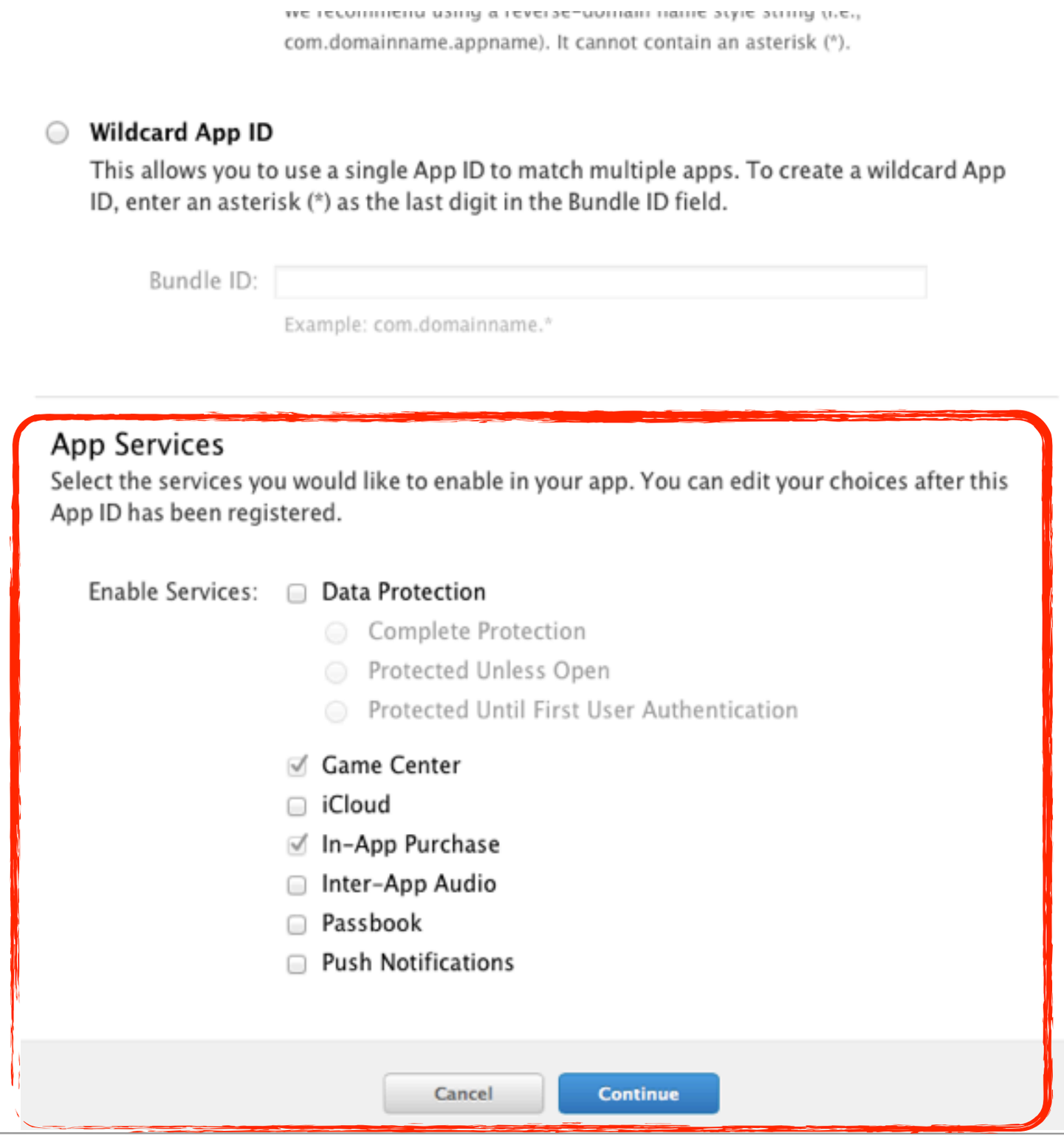
To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.

Bundle ID: de.sm-mb.BuyFruit

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).

Wildcard App ID

iTunes Connect (iTC) Metadata Setup



iTunes Connect (iTC) Metadata Setup

Registration complete.

This App ID is now registered to your account and can be used in your provisioning profiles.

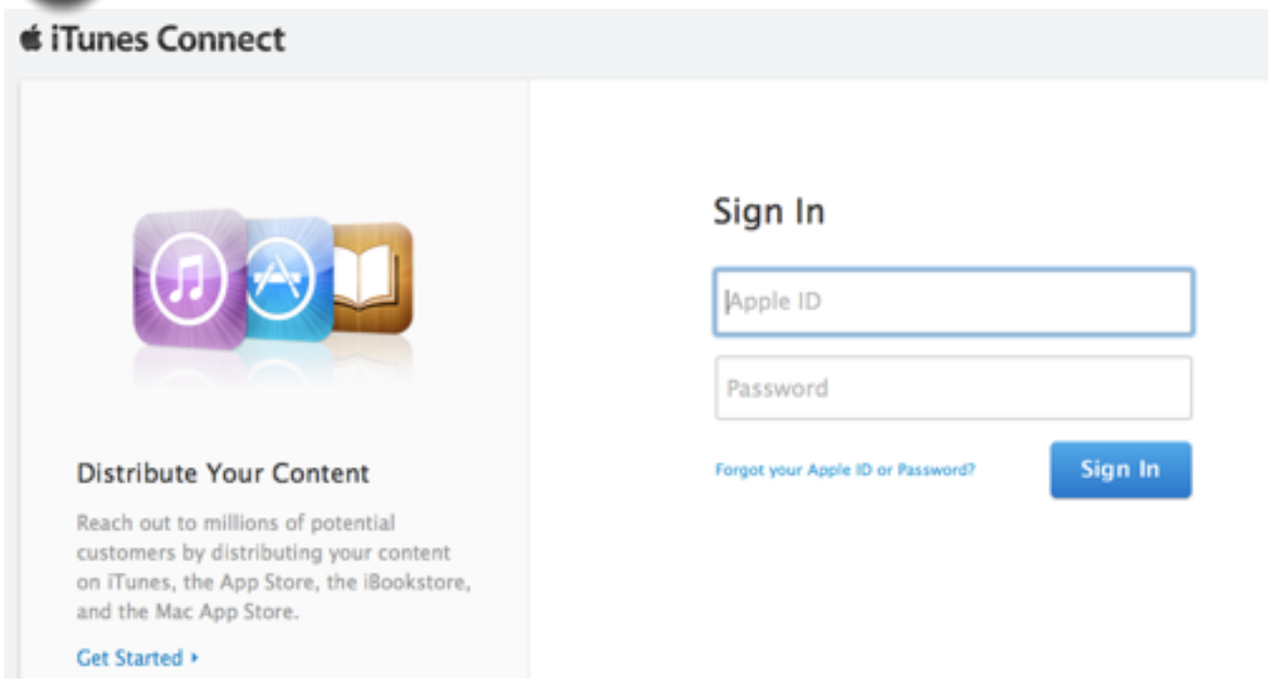
App ID Description:	**BuyFruit**
Identifier:	**FR44[redacted]9.de.sm-mb.BuyFruit**
Data Protection:	Disabled
Game Center:	**Enabled**
iCloud:	Disabled
In-App Purchase:	**Enabled**
Inter-App Audio:	Disabled
Passbook:	Disabled
Push Notifications:	Disabled

iTunes Connect (iTC) Metadata Setup

- Create an In-App Purchase item
  - In-App Purchase **type**
  - **Reference name**: only visible to developer in iTC
  - **Product ID**: reference to product, appears in financial report. Recommendation: use reverse domain name syntax.
  - **Display name**, **display description** (with localization for different languages)
  - **Pricing**: Cleared for sale, price tier
  - Screenshot for review only

iTunes Connect (iTC) Metadata Setup

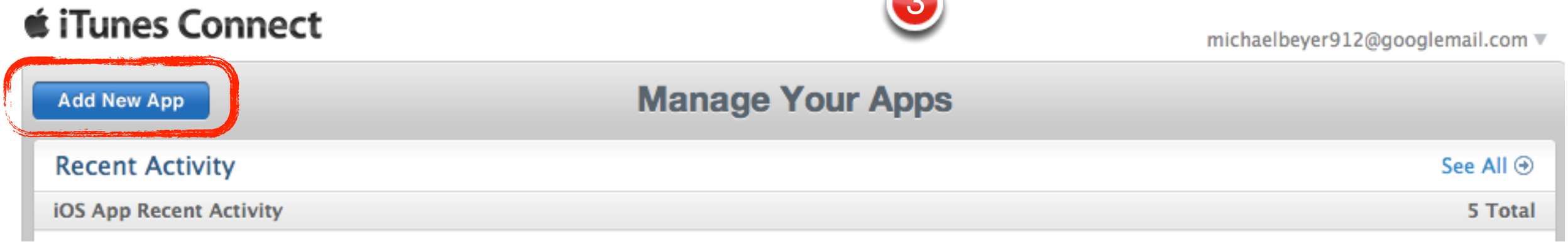
1



2

Manage Your Apps
Add, view, and manage your App Store apps.

3



iTunes Connect (iTC) Metadata Setup

1

iTunes Connect

michaelbeyer912@googlemail.com

BuyFruit

Select the availability date and price tier for your app.

Availability Date 12/Dec 31 2014

Price Tier Free

View Pricing Matrix

Discount for Educational Institutions

Custom B2B App

Unless you select specific stores, your app will be for sale in all App Stores worldwide.

Go Back Continue

2

Version Number 1.0

Copyright Michael Beyer

Category

Use the App Store Category Definitions to choose the most appropriate category for your apps.

Primary Category Food & Drink

Secondary Category (Optional) Select

Rating

For each content description, choose the level of frequency that best describes your app based on App Rating Detail.

Apps must not contain any obscene, pornographic, offensive or defamatory content or materials of any kind (text, graphics, images, photographs, etc.), or other content or materials that in Apple's reasonable judgment may be found objectionable.

Apple Content Descriptions	None	Infrequent/Mild	Frequent/Intense
Cartoon or Fantasy Violence	●	○	○

4+ App Rating

3

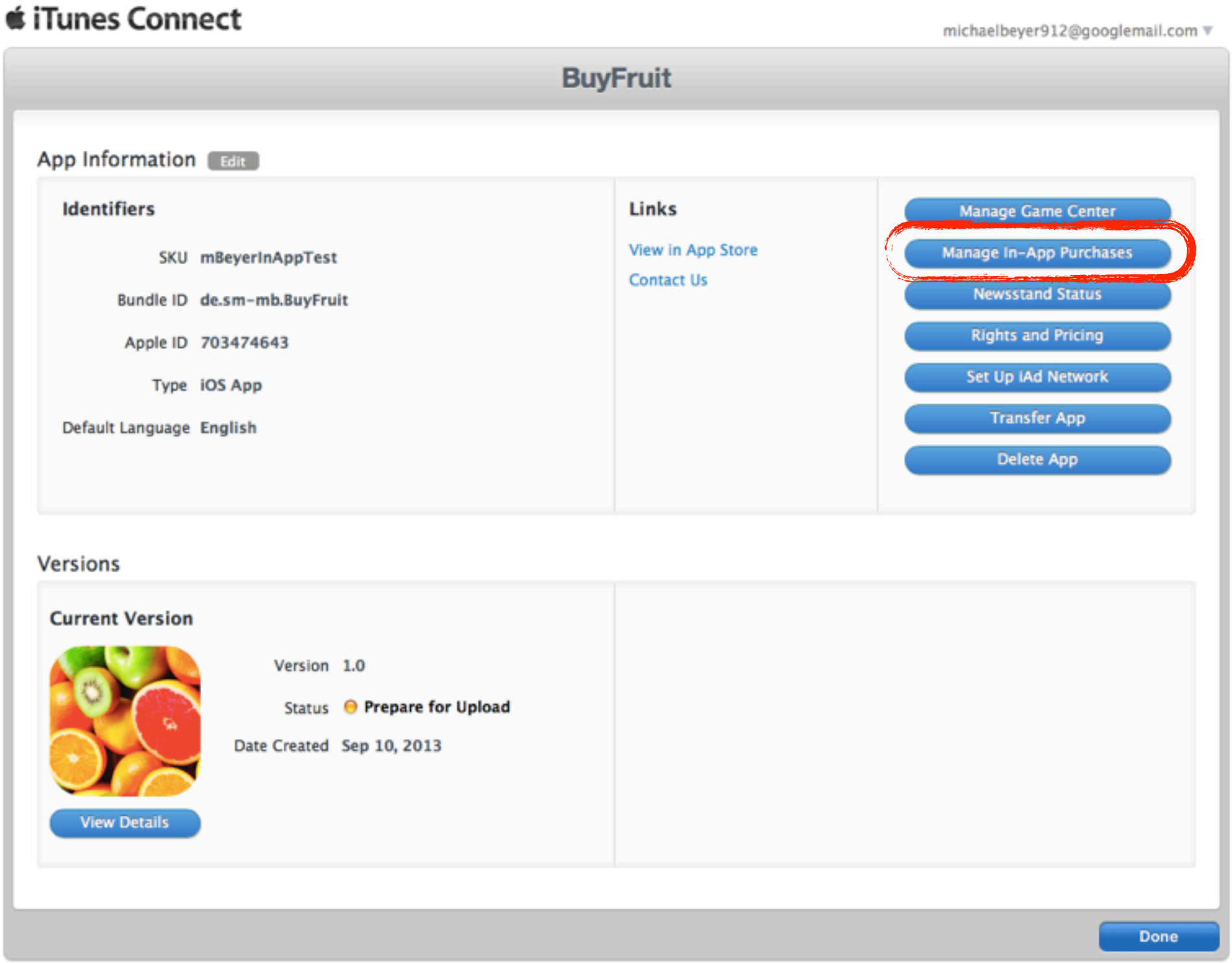
Uploads

Large App Icon

Choose File

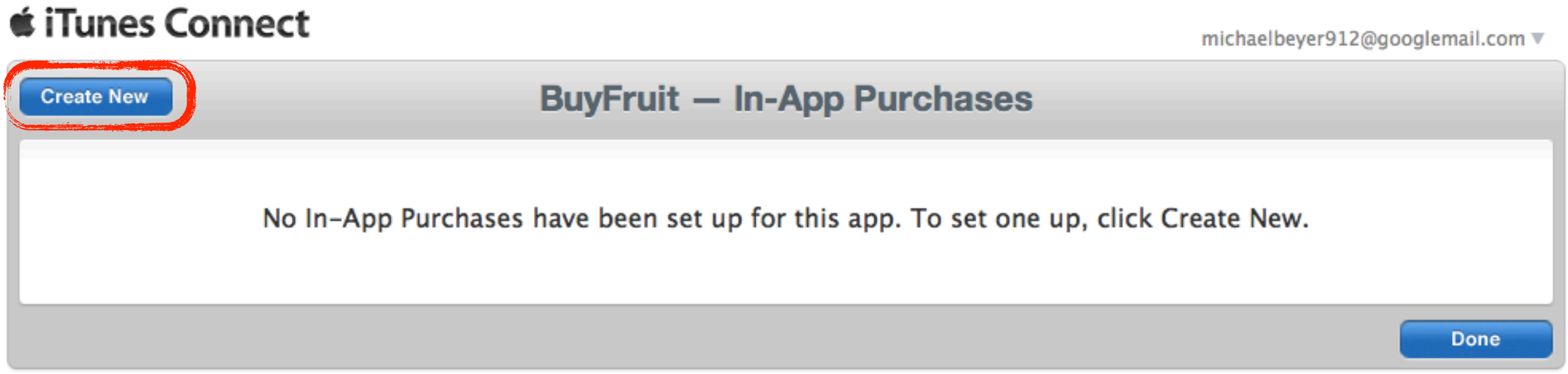
3.5-Inch Retina Display Screenshots

iTunes Connect (iTC) Metadata Setup



iTunes Connect (iTC) Metadata Setup

1



iTunes Connect (iTC) Metadata Setup

2

Select Type

Select the In-App Purchase type you want to create. If a type is missing, make sure you have agreed to all recent contracts. The Legal user must go to the "Contracts, Tax, and Banking" module on iTunes Connect to agree to the latest Paid Applications agreement. You must agree to the Developer Program License Agreement before you can access the Paid Applications agreement. To help ensure that your app is not vulnerable to fraudulent In-App Purchases, review the In-App Purchase Receipt Validation documentation.

Learn more about selling products with In-App Purchases.

Consumable

A consumable In-App Purchase must be purchased every time the user downloads it. One-time services, such as fish food in a fishing app, are usually implemented as consumables.

Select

Non-Consumable

Non-consumable In-App Purchases only need to be purchased once by users. Services that do not expire or decrease with use are usually implemented as non-consumables, such as new race tracks for a game app.

Select

Auto-Renewable Subscriptions

Auto-renewable Subscriptions allow the user to purchase updating and dynamic content for a set duration of time. Subscriptions renew automatically unless the user opts out, such as magazine subscriptions.

Select

iTunes Connect (iTC) Metadata Setup

3

BuyFruit — In-App Purchases

In-App Purchase Summary

Enter a reference name and a product ID for this In-App Purchase. You must also add at least one language, along with a display name and a description in that language.

Reference Name: Apple

Product ID: de.smmb.BuyFruit.Apple

Pricing and Availability

Enter the pricing and availability details for this In-App Purchase below.

Cleared for Sale Yes No

Price Tier Tier 1

View Pricing Matrix

Price Tier 1		
App Store	Customer Price	Your Proceeds
U.S.*	US $0.99	US $0.70
Mexico	MX	MX

iTunes Connect (iTC) Metadata Setup

4

In-App Purchase Details

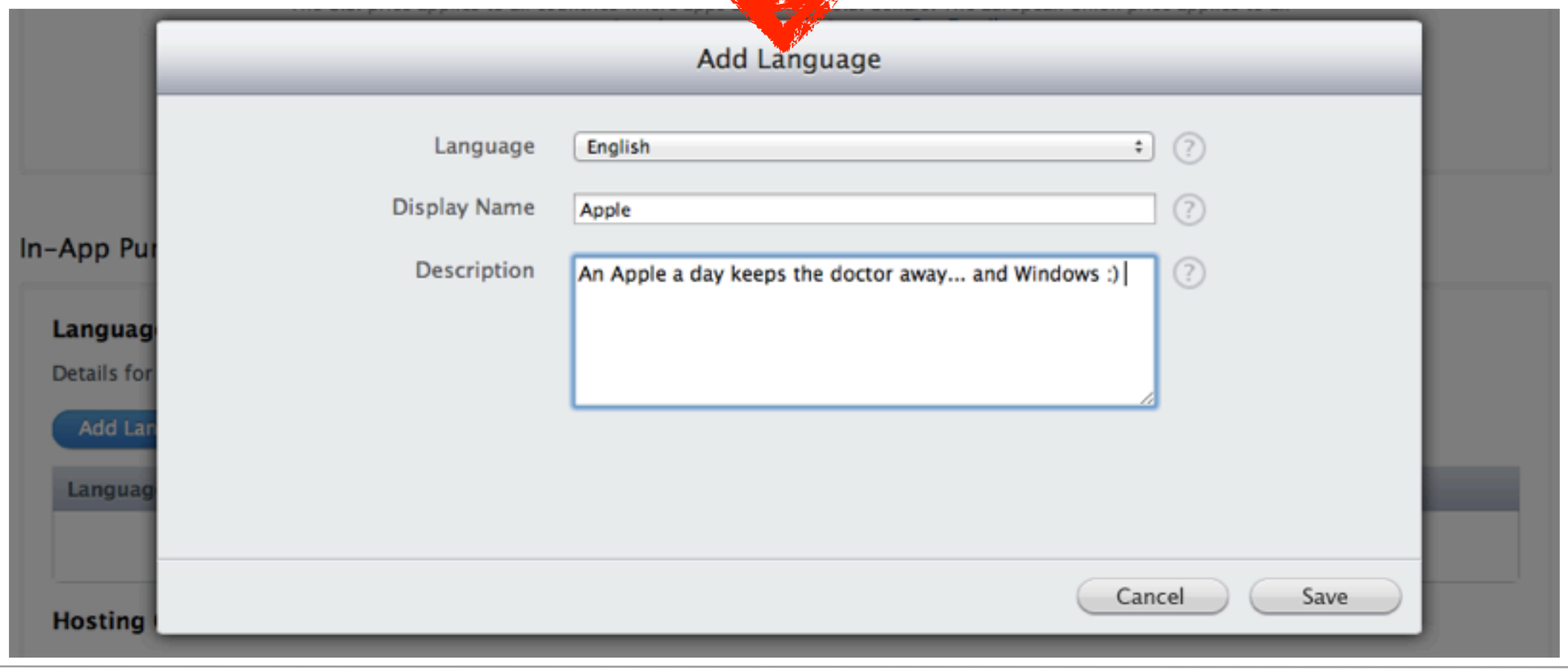
Language

Details for this In-App Purchase are shown below. You must provide at least one language at all times.

Add Language

Language	Display Name	Description	
Click Add Language to get started.			

5



iTunes Connect (iTC) Metadata Setup

6

iTunes Connect

michaelbeyer912@googlemail.com

Create New

BuyFruit — In-App Purchases

BuyFruit

Apple ID : 703474643

Bundle ID : de.sm-mb.BuyFruit

The first In-App Purchase for an app must be submitted for review at the same time that you submit an app version. You must do this on the Version Details page. Once your binary has been uploaded and your first In-App Purchase has been submitted for review, additional In-App Purchases can be submitted using the table below.

5 In-App Purchases

Search

Reference Name	Product ID	Type	Apple ID	Status
Apple	de.smmb.BuyFruit.Apple	Non-Consumable	703514772	Ready to Submit
Blackberry	de.smmb.BuyFruit.Blackberry	Non-Consumable	703522270	Ready to Submit
Orange	de.smmb.BuyFruit.Orange	Non-Consumable	703522005	Ready to Submit
Pear	de.smmb.BuyFruit.Pear	Non-Consumable	703514840	Ready to Submit
Tomato	de.smmb.BuyFruit.Tomato	Non-Consumable	703521992	Ready to Submit

View or generate a shared secret

Done

Agenda

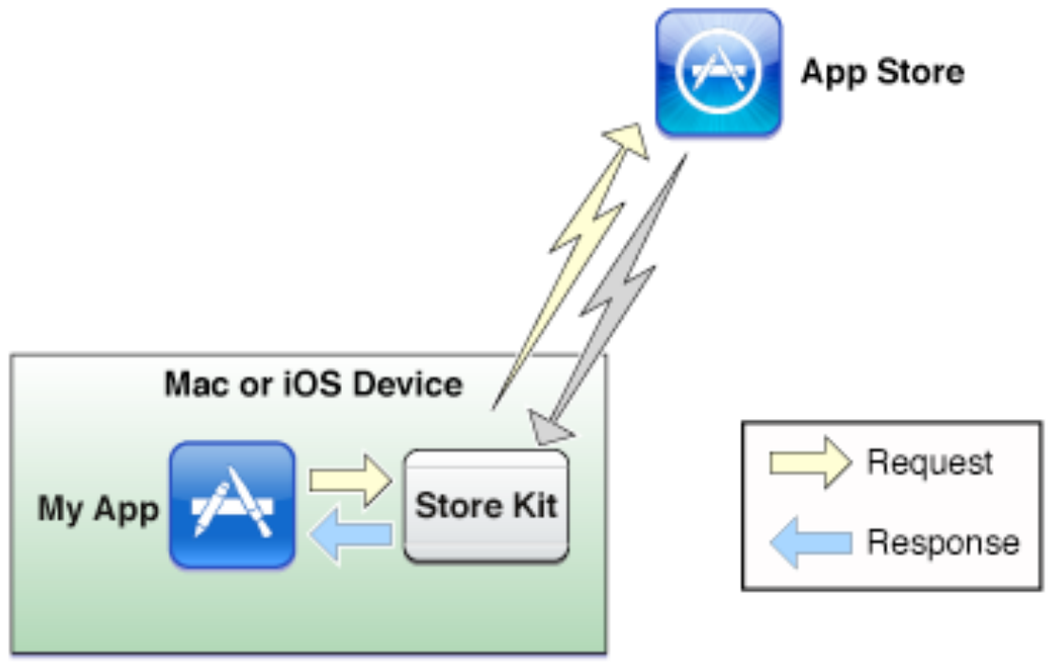
- Motivation / Examples (in) applications
- Overview
  - What is an In-App Purchase?
  - Overview: types of In-App Purchases
- **Implementation**
  - iTunes Connect Metadata Setup
  - **StoreKit on iOS (+ exercise)**
  - Testing in Sandbox
- Summary

Agenda

- Motivation / Examples (in) applications
- Overview
- **Implementation**
  - iTunes Connect Metadata Setup
  - **StoreKit on iOS (+ exercise)**
    - **StoreKit API overview**
    - Getting product information
    - Purchasing
    - Restoring completed transactions
    - Receipts
  - Testing in Sandbox
- Summary

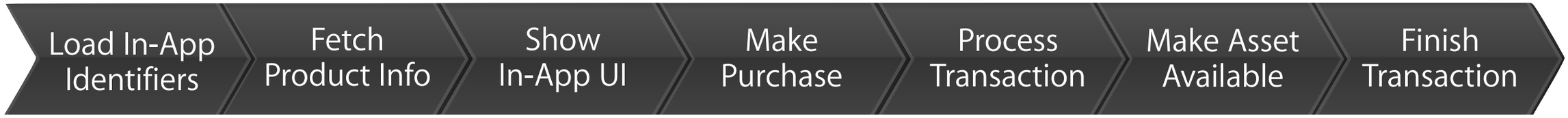
StoreKit API

- StoreKit **communicates with the AppStore** on behalf of your application: **prompts for payment** and **securely authorizes transaction**.
- Your application uses StoreKit to **receive information** from the AppStore about products you want to offer in your application. Your **application displays this information to users and allows them to purchase items.**



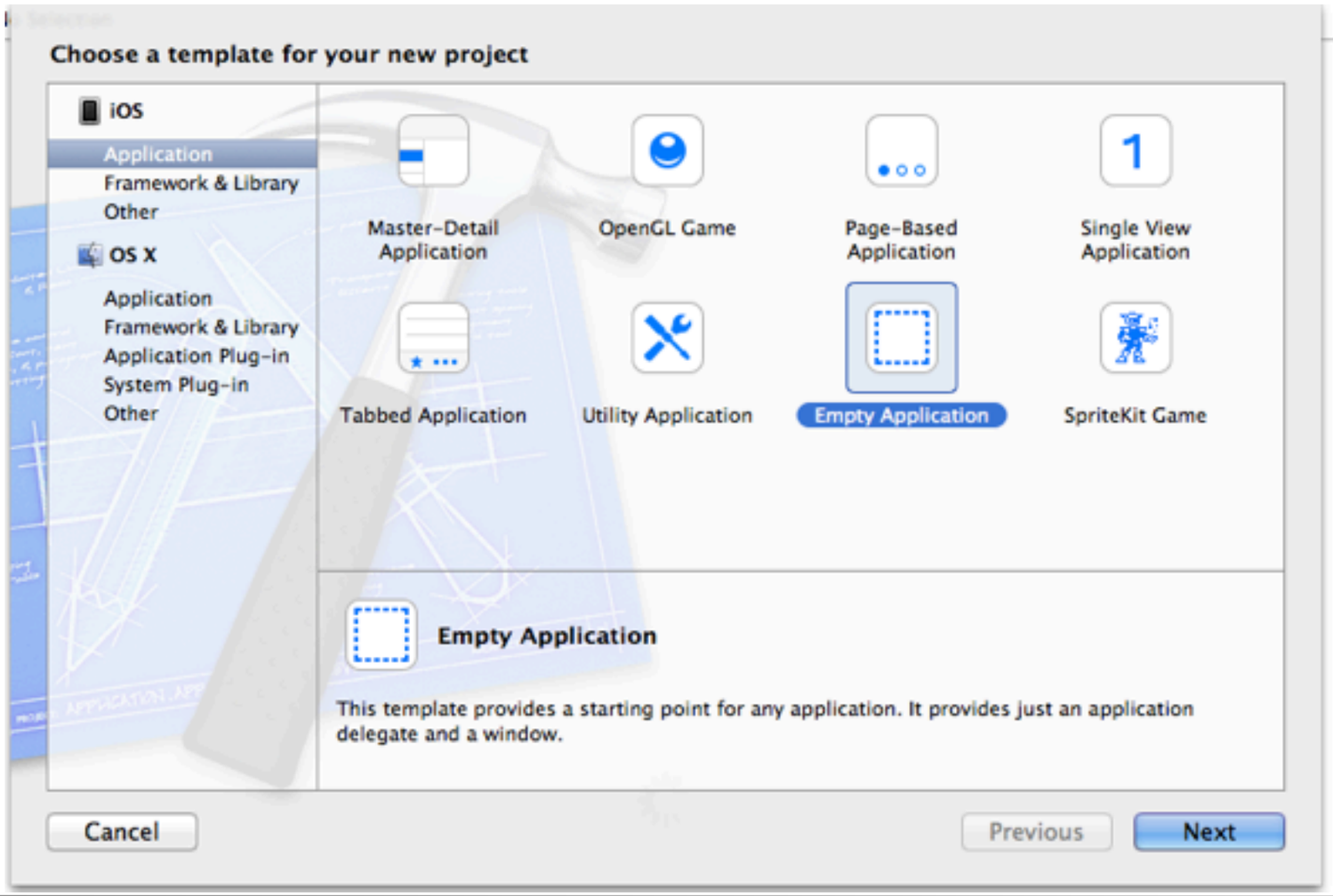
StoreKit API overview

- **StoreKit is the In-App Purchase Payment System**
  - Manages transactions for in-app purchases



Xcode Setup

- Create a new iOS project in Xcode, select the „Empty Application“ template and name the project BuyFruit.
- The Bundle ID of you app MUST match the one in iTC!

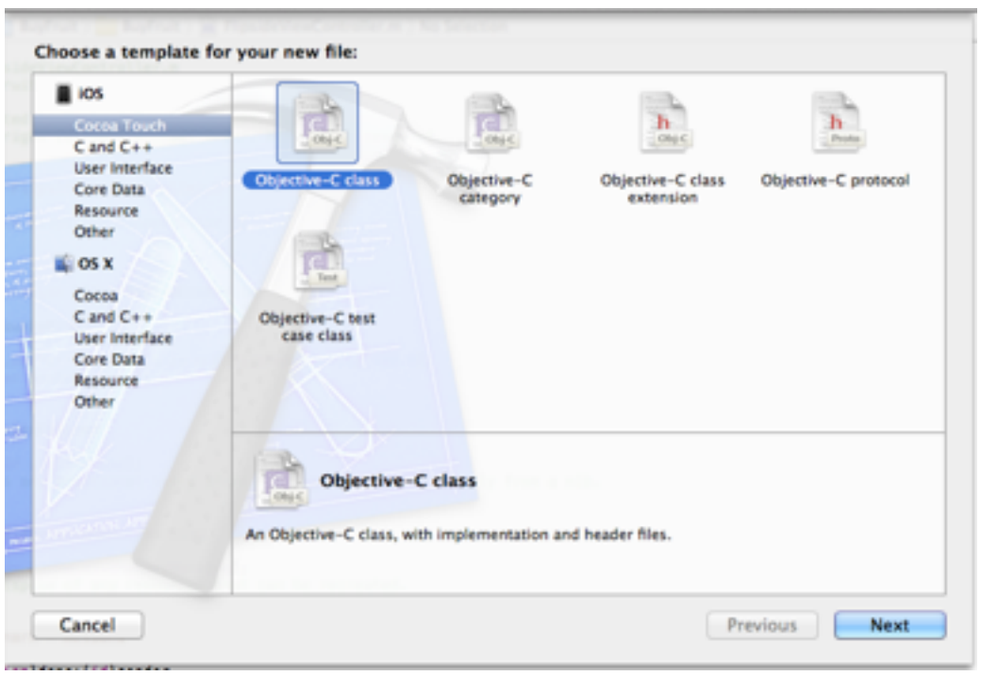


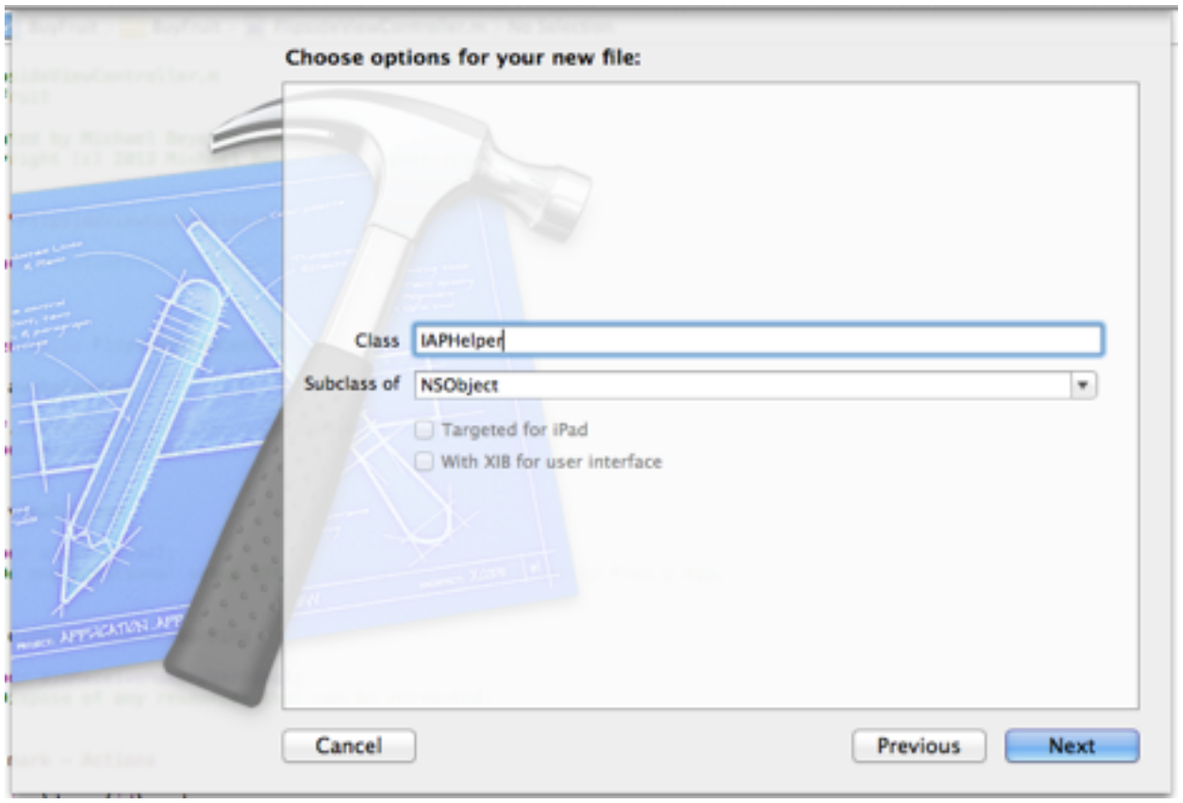
Xcode Setup

- We need to add StoreKit to our project.
- Select your project, select the „capabilities“ tab and go to the „In-App Purchase“ tab and turn it on!

IAPHelper class

- In Xcode, create the `IAPHelper` helper class, make it a subclass of `NSObject`
- We write the `IAPHelper` class so that you can easily **subclass it for your own app, specifying the product identifiers** for your app.





Agenda

- Motivation / Examples (in) applications
- Overview
- **Implementation**
  - iTunes Connect Metadata Setup
  - **StoreKit on iOS (+ exercise)**
    - StoreKit API overview
    - **Getting product information**
    - Purchasing
    - Restoring completed transactions
    - Receipts
  - Testing in Sandbox
- Summary

Getting product information

Load In-App Identifiers | **Fetch Product Info** | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

Xcode Setup

IAPHelper.h

RequestProductsCompletionHandler is a block that „returns“ void

```
// Block definition
typedef void (^RequestProductsCompletionHandler)(BOOL success, NSArray * products);
```

We’re declaring a variable “RequestProductsCompletionHandler”. The “^” declares this to be a block.

The block takes two arguments, a Boolean and a NSArray

- Blocks will let you define actions that take place in response to an event.
- All of the code for a feature is in one place (right where you set up the listener for that event), which makes maintenance easier. This can save a mess of code and make your application much more organized.
- It makes the purpose of this section of code very clear to an observer.

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

Xcode Setup

IAPHelper.h

```
@interface IAPHelper : NSObject

- (id)initWithProductIdentifiers:(NSSet *)productIdentifiers;

- (void)requestProductsWithCompletionHandler:
(RequestProductsCompletionHandler)completionHandler;

@end
```

Initializer that takes a list of product identifiers, such as com.domain.AppName.ProductName, etc.

Method to retrieve information about the products from iTunes Connect

Xcode Setup

IAPHelper.m

```
#import "IAPHelper.h"
@import StoreKit;
```

You need to use StoreKit to access the In-App Purchase APIs, so you import the StoreKit here.

```
@interface IAPHelper () <SKProductsRequestDelegate>
@end
```

To receive a list of products from StoreKit, you need to implement the SKProductsRequestDelegate protocol.
Here you mark the class as implementing this protocol in the class extension.
We can do this in the .m file, since only the implementation cares.

```
@implementation IAPHelper {
    SKProductsRequest *_productsRequest;

    RequestProductsCompletionHandler _completionHandler;
    NSSet *_productIdentifiers;
    NSMutableSet *_purchasedProductIdentifiers;
}
```

Create an instance variable to store the SKProductsRequest you will issue to retrieve a list of products

Keep track of the completion handler for the outstanding products request, the list of product identifiers passed in, and the list of product identifiers that have been previously purchased.

Xcode Setup

IAPHelper.m

```
- (id)initWithProductIdentifiers:(NSSet *)productIdentifiers
{
    self = [super init];
    if (self) {
        _productIdentifiers = productIdentifiers;

        _purchasedProductIdentifiers = [NSMutableSet set];
        for (NSString * productIdentifier in _productIdentifiers) {
            BOOL productPurchased = [[NSUserDefaults standardUserDefaults]
boolForKey:productIdentifier];
            if (productPurchased) {
                [_purchasedProductIdentifiers addObject:productIdentifier];
                NSLog(@"Previously purchased: %@", productIdentifier);
            } else {
                NSLog(@"Not purchased: %@", productIdentifier);
            }
        }
    }
    return self;
}
```

Store product identifiers

Check for previously purchased products

Xcode Setup

Getting product information – Product Request

IAPHelper.m

```
- (void)requestProductsWithCompletionHandler:
(RequestProductsCompletionHandler)completionHandler
{
    _completionHandler = [completionHandler copy];
```

A copy of the completion handler block inside the instance variable so it can notify the caller when the product request completes.

```
    _productsRequest = [[SKProductsRequest alloc]
initWithProductIdentifiers:_productIdentifiers];

    _productsRequest.delegate = self;
    [_productsRequest start];
}
```

Create a new instance of SKProductsRequest, which is the Apple-written class that contains the code to pull the info from iTunes Connect

Xcode Setup

IAPHelper.m

Getting product information – Product Response – Success

```
- (void)productsRequest:(SKProductsRequest *)request didReceiveResponse:
(SKProductsResponse *)response
{
    NSLog(@"Loaded list of products...");
    _productsRequest = nil;

    NSArray * skProducts = response.products;
    for (SKProduct * skProduct in skProducts) {
        NSLog(@"Found product: %@ - Product: %@ - Price: %0.2f",
skProduct.productIdentifier, skProduct.localizedTitle, skProduct.price.floatValue);
    }

    _completionHandler(YES, skProducts);
    _completionHandler = nil;
}
```

We set the _productsRequest instance variable back to nil because we‘re done with it

On success, we log out some information about the returned products, such as the product identifier, localized title, and price.

Call the _completionHandler and set it to nil afterwards, so it doesn't get called twice

Load In-App Identifiers | **Fetch Product Info** | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

Xcode Setup

Getting product information – Product Response – Failure

IAPHelper.m

```
- (void)request:(SKRequest *)request didFailWithError:(NSError *)error
{
    UIAlertView *message = [[UIAlertView alloc] initWithTitle:@"Failed to load list
of products."
                                                      message:nil
                                                     delegate:nil
                                            cancelButtonTitle:@"OK"
                                            otherButtonTitles:nil];

    [message show];

    NSLog(@"Failed to load list of products.");

    _productsRequest = nil;

    _completionHandler(NO, nil);
    _completionHandler = nil;
}
```

We set the _productsRequest instance variable back to nil because we‘re done with it and call the completion handler and also set it to nil afterwards

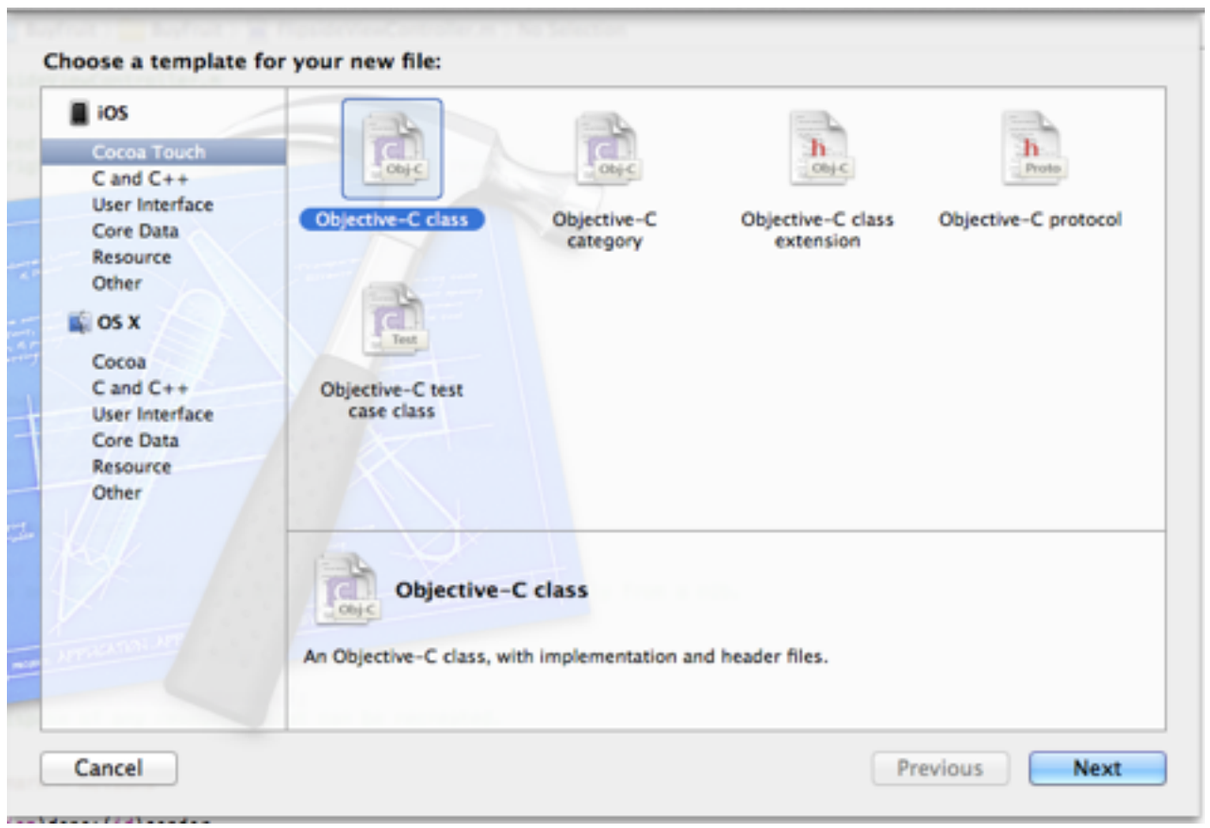
Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

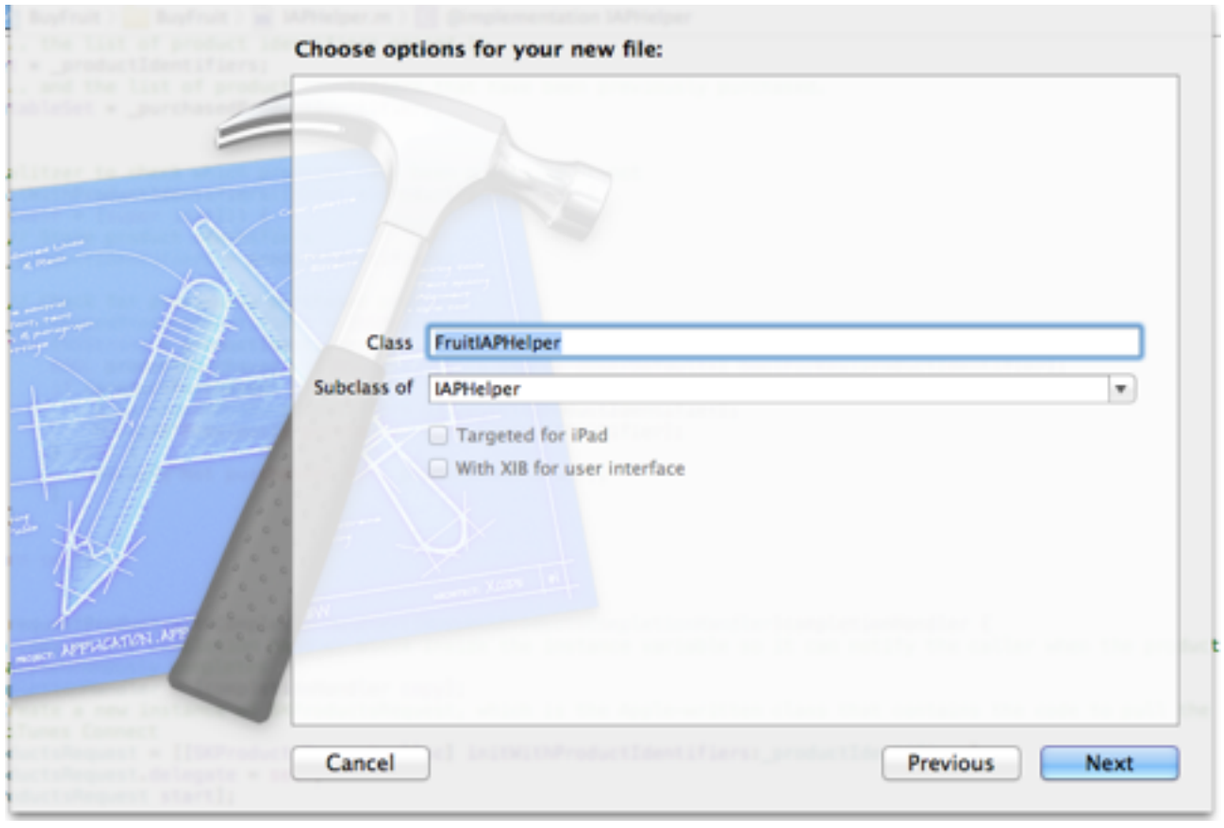
Load In-App Identifiers

- A lot of people recommend that you pull the **list of product identifiers from a web server** along with other information so you can add new In-App purchases dynamically rather than requiring an app update.
- This is true and definitely recommended, but for the purposes of this tutorial we are going to **keep things simple and just hard-code in the product identifiers** for this app.

Xcode Setup – Subclassing for your App

- In Xcode, create the `FruitIAPHelper` helper class
- Make it a subclass of `IAPHelper`





Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

Xcode Setup – Subclassing for your App

FruitIAPHelper.h

Hardcoded, local, product identifiers

```
#import "IAPHelper.h"

@interface FruitIAPHelper : IAPHelper

+ (FruitIAPHelper *)sharedInstance;

@end
```

defines a static method to return the single, global instance of this class

Xcode Setup – Subclassing for your App

FruitIAPHelper.m

Hardcoded, local, product identifiers

```
#import "FruitIAPHelper.h"

static NSString *kIdentifierApple       = @"de.tum.in.www1.sgdws13.BuyFruit.Apple";
static NSString *kIdentifierBlackberry  = @"de.tum.in.www1.sgdws13.BuyFruit.Blackberry";
static NSString *kIdentifierOrange      = @"de.tum.in.www1.sgdws13.BuyFruit.Orange";
static NSString *kIdentifierPear        = @"de.tum.in.www1.sgdws13.BuyFruit.Pear";
static NSString *kIdentifierTomato      = @"de.tum.in.www1.sgdws13.BuyFruit.Tomato";

@implementation FruitIAPHelper

+ (FruitIAPHelper *)sharedInstance {
    static FruitIAPHelper *sharedInstance;
    static dispatch_once_t once;
    dispatch_once(&once, ^{

        NSSet *productIdentifiers = [NSSet setWithObjects:
                                     kIdentifierApple,
                                     kIdentifierBlackberry,
                                     kIdentifierOrange,
                                     kIdentifierPear,
                                     kIdentifierTomato,
                                     nil];
        sharedInstance = [[self alloc] initWithProductIdentifiers:productIdentifiers];
    });
    return sharedInstance;
}
```

implements the Singleton pattern in Objective-C

make it thread safe

returns a single, global instance of the class.

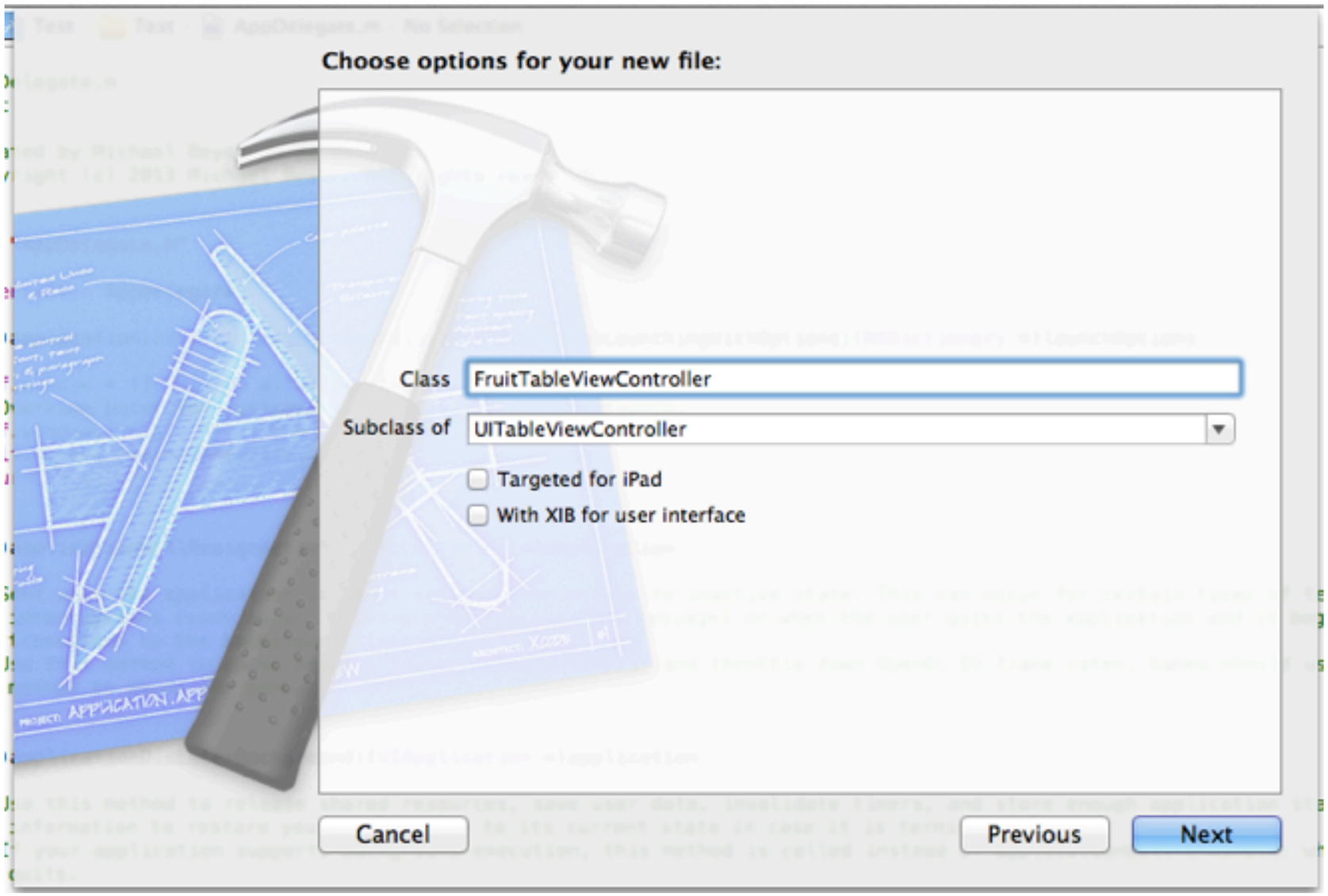
calls the superclasses initializer to pass in all the productIdentifiers that we created in iTunes Connect

Displaying The Products

Load In-App Identifiers → Fetch Product Info → **Show In-App UI** → Make Purchase → Process Transaction → Make Asset Available → Finish Transaction

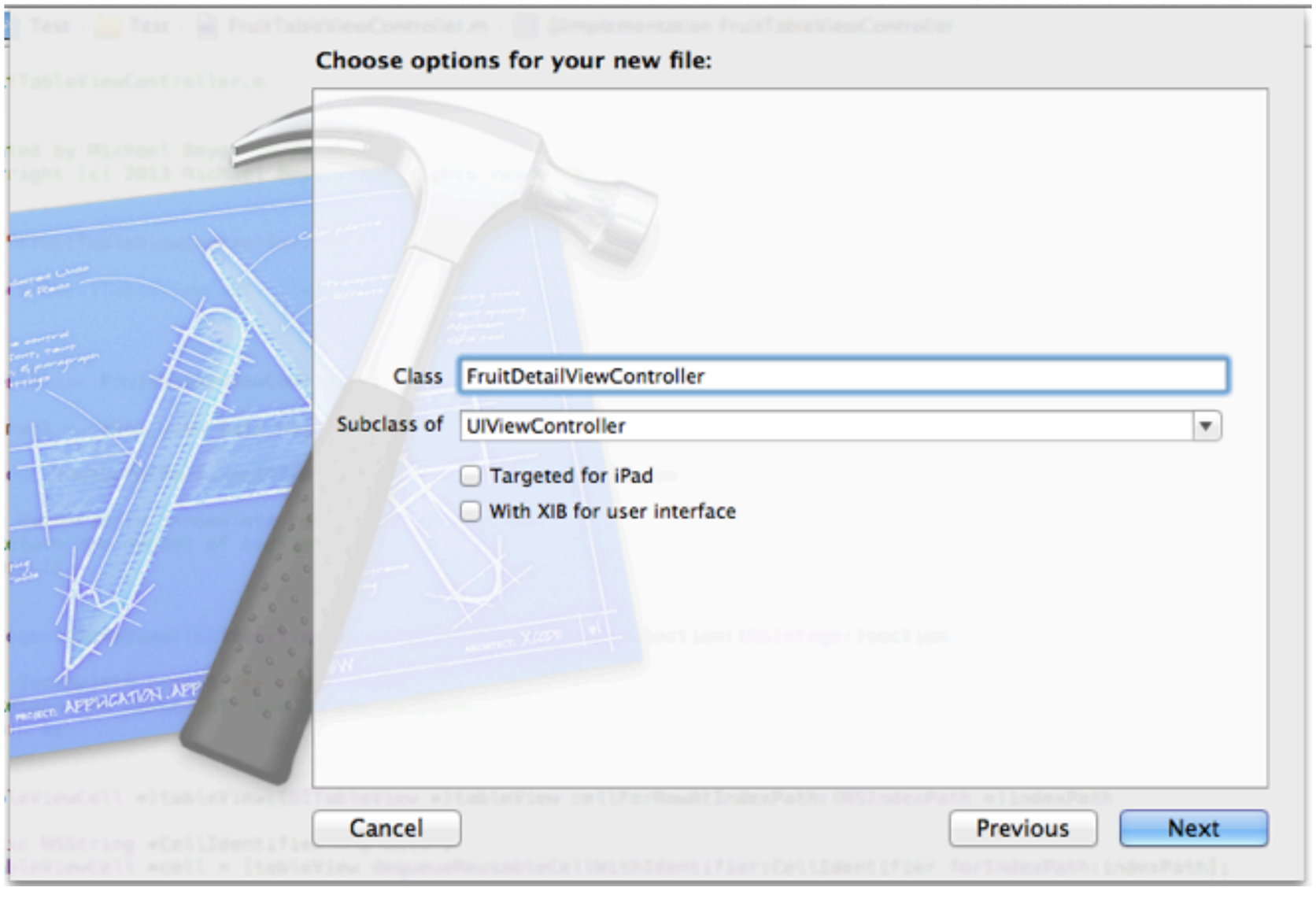
User Interface

- Create a new class name it `FruitTableViewController` and make it a subclass of `UITableViewController`



User Interface

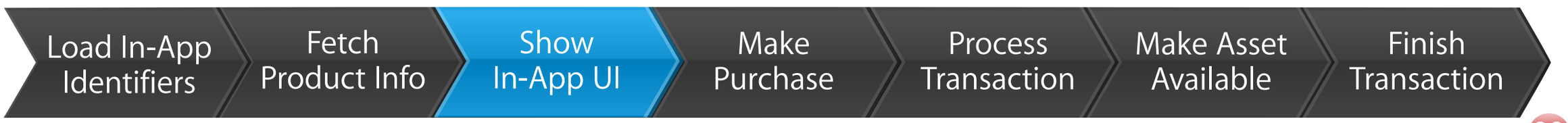
- Create a new class name it `FruitDetailViewController` and make it a subclass of `UIViewController`



User Interface

FruitTableViewController.h

```
@import UIKit;
#import "FruitDetailViewController.h"

@interface FruitTableViewController : UITableViewController

@property (nonatomic, strong) FruitDetailViewController *fruitDetailViewController;

@end
```



User Interface

FruitTableViewController.m

```
#import "FruitTableViewController.h"
#import "FruitIAPHelper.h"

@interface FruitTableViewController ()

@property (nonatomic, strong) NSArray *products;
@property (nonatomic, strong) NSNumberFormatter *priceFormatter;

@end

@implementation FruitTableViewController
```

Allows you to flexibly convert backwards and forwards between numbers (NSNumber) and localized string (NSString) representations of those numbers

User Interface

FruitTableViewController.m

```
- (id)init
{
    self = [super init];
    if (self) {
        self.title = @"Buy Fruits";

        self.priceFormatter = [[NSNumberFormatter alloc] init];
        [self.priceFormatter setFormatterBehavior:NSNumberFormatterBehavior10_4];
        [self.priceFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];
    }
    return self;
}
```

Load In-App Identifiers | Fetch Product Info | **Show In-App UI** | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

User Interface

FruitTableViewController.m

```
- (void)viewWillAppear:(BOOL)animated
{
    [self reload];
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(productPurchased:) name:IAPHelperProductPurchasedNotification
object:nil];
    [super viewWillAppear:animated];

    self.refreshControl = [[UIRefreshControl alloc] init];
    [self.refreshControl addTarget:self action:@selector(reload)
forControlEvents:UIControlEventValueChanged];
    [self.refreshControl beginRefreshing];
}

- (void)viewWillDisappear:(BOOL)animated
{
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

User Interface

FruitTableViewController.m

```
#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Return the number of sections.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section
{
    // Return the number of rows in the section.
    return [self.products count];
}
```

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

User Interface

FruitTableViewController.m

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

    // Configure the cell...
    if (cell == nil) {
        cell = [[UITableViewCell alloc]initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:CellIdentifier];
    }

    SKProduct *product = self.products[indexPath.row];
    cell.textLabel.text = product.localizedTitle;
    cell.detailTextLabel.text = [product.price stringValue];

    [self.priceFormatter setLocale:product.priceLocale];
    cell.detailTextLabel.text = [self.priceFormatter stringFromNumber:product.price];

    if ([[FruitIAPHelper sharedInstance] productPurchased:product.productIdentifier]) {
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
        cell.accessoryView = nil;
    } else {
        UIButton *buyButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
        buyButton.frame = CGRectMake(0, 0, 72, 37);
        [buyButton setTitle:@"Buy" forState:UIControlStateNormal];
        buyButton.tag = indexPath.row;
        [buyButton addTarget:self action:@selector(buyButtonTapped:)
forControlEvents:UIControlEventTouchUpInside];
        cell.accessoryType = UITableViewCellAccessoryNone;
        cell.accessoryView = buyButton;
    }
    return cell;
}
```

User Interface

FruitTableViewController.m

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath
{
    // Gives the product back
    SKProduct *product = self.products[indexPath.row];

    // Creates the DetailViewController
    self.fruitDetailViewController = [[FruitDetailViewController alloc]init];
    self.fruitDetailViewController.product = product;

    // Pushes the DetailViewController
    [self.navigationController pushViewController:self.fruitDetailViewController
animated:YES];
}
```

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | Process Transaction | Make Asset Available | Finish Transaction

User Interface

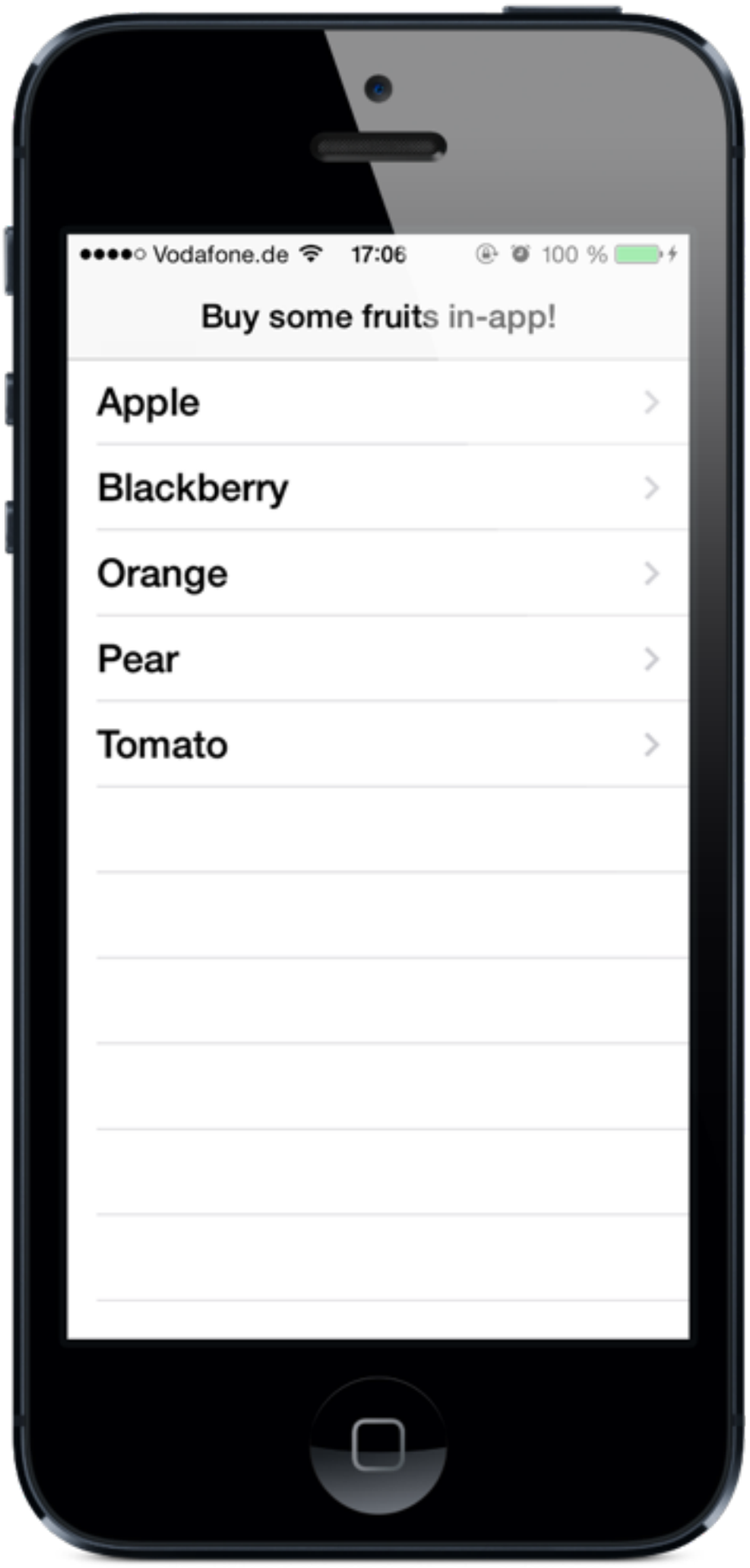
FruitTableViewController.m

```
#pragma mark - StoreKit
- (void)reload {
    self.products = nil;
    [self.tableView reloadData];
    [[FruitIAPHelper sharedInstance] requestProductsWithCompletionHandler:^(BOOL
success, NSArray *products) {
        if (success) {
            self.products = products;
            [self.tableView reloadData];
        }
        [self.refreshControl endRefreshing];
    }];
}

- (void)productPurchased:(NSNotification *)notification {
    NSString * productIdentifier = notification.object;
    [self.products enumerateObjectsUsingBlock:^(SKProduct * product, NSUInteger
idx, BOOL *stop) {
        if ([product.productIdentifier isEqualToString:productIdentifier]) {
            [self.tableView reloadRowsAtIndexPaths:@[[NSIndexPath
indexPathForRow:idx inSection:0]] withRowAnimation:UITableViewRowAnimationFade];
            *stop = YES;
        }
    }];
}
```

User Interface

FruitTableViewController.m

```
- (void)buyButtonTapped:(id)sender
{
    UIButton *buyButton = (UIButton *)sender;
    SKProduct *product = self.products[buyButton.tag];
    NSLog(@"Buying %@...", product.productIdentifier);
    [[FruitIAPHelper sharedInstance] buyProduct:product];
}
```


Vodafone.de
17:06
100 %
Buy some fruits in-app!
Apple
Blackberry
Orange
Pear
Tomato

Agenda

- Motivation / Examples (in) applications
- Overview
- **Implementation**
  - iTunes Connect Metadata Setup
  - **StoreKit on iOS (+ exercise)**
    - StoreKit API overview
    - Getting product information
    - **Purchasing**
    - Restoring completed transactions
    - Receipts
  - Testing in Sandbox
- Summary

Purchasing

- Making the **purchase** (create a `SKPayment` object and specify what product identifier the user wants to purchase)
- **Observer** gets called (Apple ID / PW, charging account, send a success or failure)
- Make **content available to user** (in our case just setting that flag in `NSUserDefaults` and storing it in the purchasedProducts array)
- Complete the transaction

Purchasing

- Once again, most of this is going to be in the IAPHelper class for easy reuse.
- Start by making the following changes to `IAPHelper.h` ...

Purchasing

IAPHelper.h

```
@import Foundation;
@import StoreKit;

UIKIT_EXTERN NSString *const IAPHelperProductPurchasedNotification;

typedef void (^RequestProductsCompletionHandler)(BOOL success, NSArray * products);

@interface IAPHelper : NSObject

// Method definition
- (id)initWithProductIdentifiers:(NSSet *)productIdentifiers;
- (void)requestProductsWithCompletionHandler:
(RequestProductsCompletionHandler)completionHandler;

- (void)buyProduct:(SKProduct *)product;

- (BOOL)productPurchased:(NSString *)productIdentifier;
```

Declaration of a notification we'll use to notify listeners when a product has been purchased

Method to start buying a product

Method to determine if a product has been purchased

Purchasing

IAPHelper.m

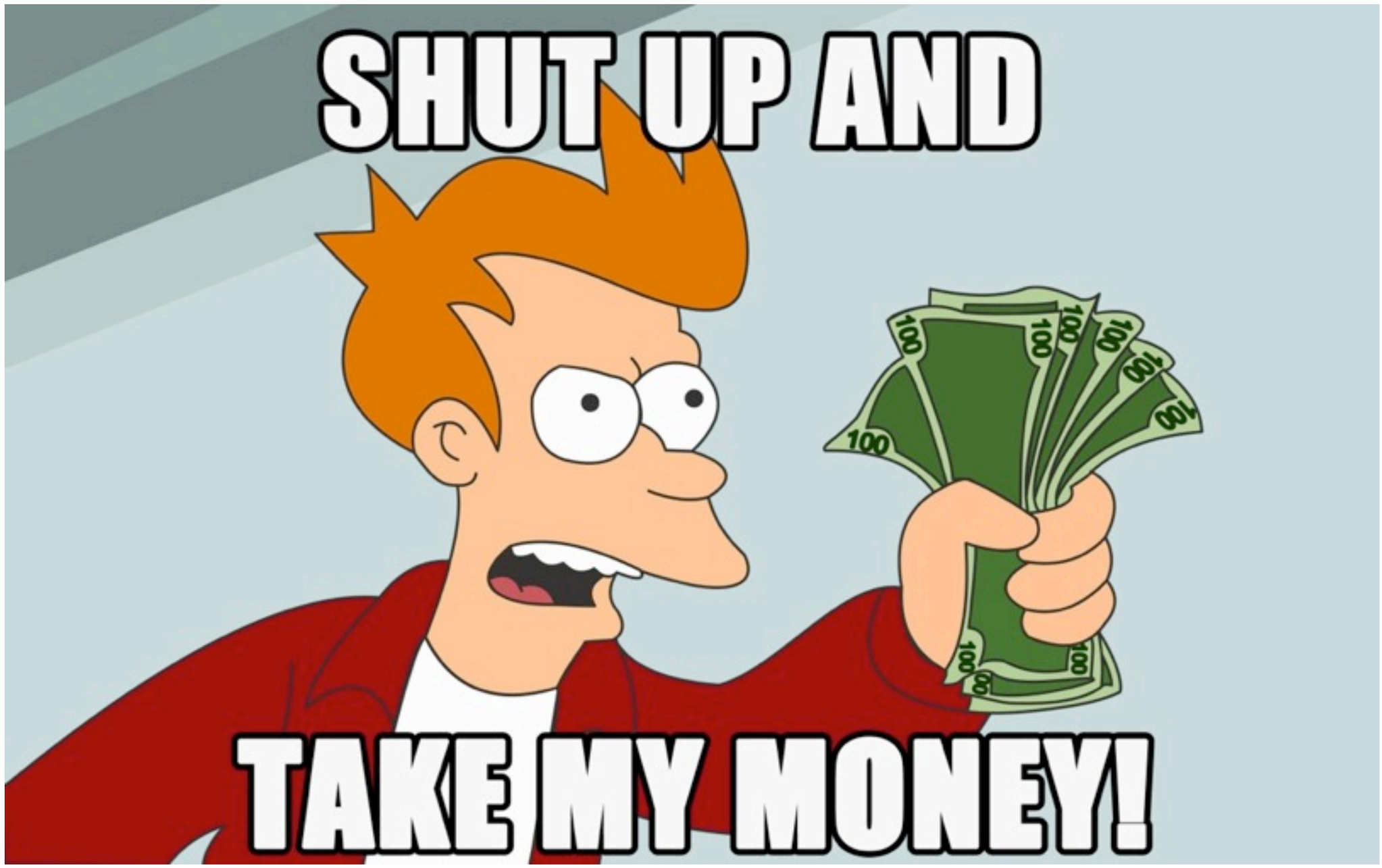
```
- (BOOL)productPurchased:(NSString *)productIdentifier {
    return [_purchasedProductIdentifiers containsObject:productIdentifier];
}
```

make sure the product is allowed to be purchased

```
- (void)buyProduct:(SKProduct *)product {
    NSLog(@"Buying %@...", product.productIdentifier);

    SKPayment * payment = [SKPayment paymentWithProduct:product];
    [[SKPaymentQueue defaultQueue] addPayment:payment];
}
```

marks the purchase as in-progress, and issue an SKPayment to the SKPaymentQueue.

Load In-App Identifiers | Fetch Product Info | Show In-App UI | **Make Purchase** | Process Transaction | Make Asset Available | Finish Transaction



That’s literally all it takes to let the user send you real cash! :-)

Purchasing

- However, if you're letting your users give you money, you better give them something good in return!
- So you need to add some code to identify when a payment "transaction" has finished, and process it accordingly.
- Make the following changes to `IAPHelper.h` ...

Purchasing

IAPHelper.m

```
@interface IAPHelper () <SKProductsRequestDelegate, SKPaymentTransactionObserver>
```

modify the class extension to mark the class as implementing the SKPaymentTransactionObserver

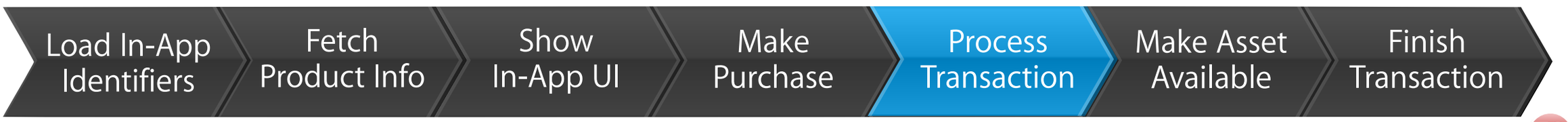
```
...

- (id)initWithProductIdentifiers:(NSSet *)productIdentifiers {
    if ((self = [super init])) {
        ....
         for (NSString * productIdentifier in _productIdentifiers) {
             BOOL productPurchased = [[NSUserDefaults standardUserDefaults]
boolForKey:productIdentifier];
             if (productPurchased) {
                 ...
             }
         }
         [[SKPaymentQueue defaultQueue] addTransactionObserver:self];
    }
    return self;
```

Add self as transaction observer

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | **Process Transaction** | Make Asset Available | Finish Transaction

Purchasing

- There’s one really important thing about this:
- It could quite possibly happen that a user starts a purchase (and gets charged for it), but before Apple can respond with success or failure, the user suddenly loses network connection, or terminates your app.
- **The user will still expect their purchase though.**
- Luckily, there‘s a solution for that: **Apple will keep track of any purchase transactions that haven’t yet been fully processed by your app, and will notify the transaction observer about them.**



Purchasing

AppDelegate.m

```
#import "AppDelegate.h"
// register our class as a transaction observer
#import "FruitIAPHelper.h"

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions
{
    [FruitIAPHelper sharedInstance];
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    [self.window makeKeyAndVisible];

   ...

    return YES;
}
```


Purchasing

IAPHelper.m

```
- (void)paymentQueue:(SKPaymentQueue *)queue updatedTransactions:(NSArray
*)transactions {
    for (SKPaymentTransaction *transaction in transactions) {
        switch (transaction.transactionState) {
            case SKPaymentTransactionStatePurchased:
                [self completeTransaction:transaction];
                break;
            case SKPaymentTransactionStateFailed:
                [self failedTransaction:transaction];
                break;
            case SKPaymentTransactionStateRestored:
                [self restoreTransaction:transaction];
            default:
            break;
        }
    };
}
```

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | **Process Transaction** | Make Asset Available | Finish Transaction

Purchasing

IAPHelper.m

```
// called when the transaction was successful
- (void)completeTransaction:(SKPaymentTransaction *)transaction
{
    NSLog(@"completeTransaction...");

    UIAlertView *message = [[UIAlertView alloc] initWithTitle:@"Bought successfully!"
                                                      message:@"Thank you for your
purchase. Enjoy!"
                                                     delegate:nil
                                            cancelButtonTitle:@"OK"
                                            otherButtonTitles:nil];
    [message show];

    [self provideContentForProductIdentifier:transaction.payment.productIdentifier];
    [[NSUserDefaults standardUserDefaults] setBool:YES
forKey:transaction.payment.productIdentifier];
    [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
}
```

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | **Process Transaction** | Make Asset Available | Finish Transaction

Purchasing

IAPHelper.m

```
// called when a transaction has been restored and successfully completed
- (void)restoreTransaction:(SKPaymentTransaction *)transaction
{
    NSLog(@"restoreTransaction...");

    UIAlertView *message = [[UIAlertView alloc] initWithTitle:@"Restored
successfully!"
                                                      message:@"Enjoy!"
                                                     delegate:nil
                                            cancelButtonTitle:@"OK"
                                            otherButtonTitles:nil];
    [message show];

    [self
provideContentForProductIdentifier:transaction.originalTransaction.payment.product
Identifier];
    [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
}
```

Load In-App Identifiers | Fetch Product Info | Show In-App UI | Make Purchase | **Process Transaction** | Make Asset Available | Finish Transaction

Purchasing

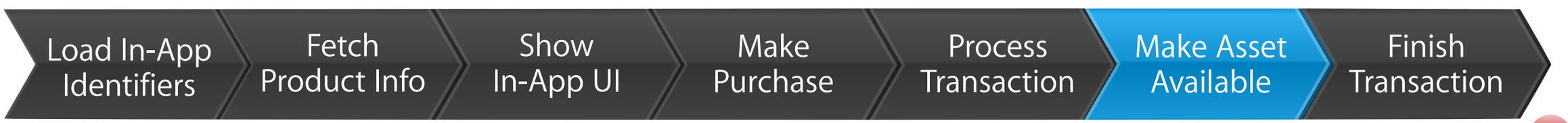
IAPHelper.m

```
// called when a transaction has failed
- (void)failedTransaction:(SKPaymentTransaction *)transaction
{
    NSLog(@"failedTransaction...");

    if (transaction.error.code != SKErrorPaymentCancelled) {
        NSLog(@"Transaction error: %@", transaction.error.localizedDescription);

        UIAlertView *message = [[UIAlertView alloc] initWithTitle:@"Ups!"
message:transaction.error.localizedDescription
                                                         delegate:nil
                                                cancelButtonTitle:@"OK"
                                                otherButtonTitles:nil];
        [message show];
    }

    [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
}
```

Purchasing

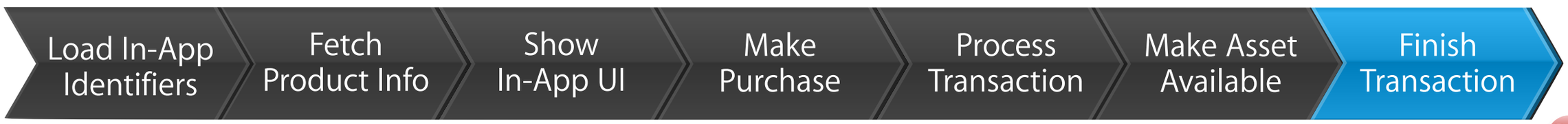
IAPHelper.m

```
- (void)provideContentForProductIdentifier:(NSString *)productIdentifier {
    NSLog(@"provideContentForProductIdentifier");

    [_purchasedProductIdentifiers addObject:productIdentifier];
    [[NSUserDefaults standardUserDefaults] setBool:YES forKey:productIdentifier];
    [[NSUserDefaults standardUserDefaults] synchronize];
    [[NSNotificationCenter defaultCenter]
postNotificationName:IAPHelperProductPurchasedNotification object:productIdentifier
userInfo:nil];
}
```



Purchasing

- That‘s it!
- Check if `[[SKPaymentQueue defaultQueue] finishTransaction: transaction];` gets called in the `completeTransaction`, `failedTransaction` and `restoreTransaction` methods.
- All we have to do now, is updating the User Interface.



Agenda

- Motivation / Examples (in) applications
- Overview
- **Implementation**
  - iTunes Connect Metadata Setup
  - **StoreKit on iOS (+ exercise)**
    - StoreKit API overview
    - Getting product information
    - Purchasing
    - **Restoring completed transactions**
    - Receipts
  - Testing in Sandbox
- Summary

Restoring completed transactions

- Getting non-consumable purchases back, on another device of the customer or if the user reinstalls your app, etc.
- If you have non-consumable or auto-renew subscription types, you must implement this feature, or Apple will reject your app.
- Don’t auto-restore on launch

Restoring completed transactions

IAPHelper.h

```
@import Foundation;
@import StoreKit;

UIKIT_EXTERN NSString *const IAPHelperProductPurchasedNotification;

typedef void (^RequestProductsCompletionHandler)(BOOL success, NSArray * products);

@interface IAPHelper : NSObject

- (id)initWithProductIdentifiers:(NSSet *)productIdentifiers;
- (void)requestProductsWithCompletionHandler:
(RequestProductsCompletionHandler)completionHandler;
- (void)buyProduct:(SKProduct *)product;
- (BOOL)productPurchased:(NSString *)productIdentifier;
- (void)restoreCompletedTransactions;

@end
```

Method to restore completed transactions

Restoring completed transactions

IAPHelper.m

```
- (void)restoreCompletedTransactions {
    [[SKPaymentQueue defaultQueue] restoreCompletedTransactions];
}
```

Restoring completed transactions

IAPHelper.m

```
- (void)paymentQueue:(SKPaymentQueue *)queue updatedTransactions:(NSArray
*)transactions
{
    for (SKPaymentTransaction *transaction in transactions) {
        switch (transaction.transactionState) {
            case SKPaymentTransactionStatePurchased:
                [self completeTransaction:transaction];
                break;
            case SKPaymentTransactionStateFailed:
                [self failedTransaction:transaction];
                break;
            case SKPaymentTransactionStateRestored:
                [self restoreTransaction:transaction];
            default:
            break;
        }
    };
}
```

Restoring completed transactions

IAPHelper.m

```
// called when a transaction has been restored and successfully completed
- (void)restoreTransaction:(SKPaymentTransaction *)transaction
{
    NSLog(@"restoreTransaction...");
    [self
provideContentForProductIdentifier:transaction.originalTransaction.payment.productI
dentifier];
    [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
}

- (void)provideContentForProductIdentifier:(NSString *)productIdentifier
{
    NSLog(@"provideContentForProductIdentifier");

    [_purchasedProductIdentifiers addObject:productIdentifier];
    [[NSUserDefaults standardUserDefaults] setBool:YES forKey:productIdentifier];
    [[NSUserDefaults standardUserDefaults] synchronize];
    [[NSNotificationCenter defaultCenter]
postNotificationName:IAPHelperProductPurchasedNotification object:productIdentifier
userInfo:nil];
}
```

Restoring completed transactions

IAPHelper.m

```
// called when a transaction has been restored and successfully completed
- (void)restoreTransaction:(SKPaymentTransaction *)transaction
{
    NSLog(@"restoreTransaction...");
    [self
provideContentForProductIdentifier:transaction.originalTransaction.payment.productI
dentifier];
    [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
}
```

```
- (void)provideContentForProductIdentifier:(NSString *)productIdentifier
{
    NSLog(@"provideContentForProductIdentifier");

    [_purchasedProductIdentifiers addObject:productIdentifier];
    [[NSUserDefaults standardUserDefaults] setBool:YES forKey:productIdentifier];
    [[NSUserDefaults standardUserDefaults] synchronize];
    [[NSNotificationCenter defaultCenter]
postNotificationName:IAPHelperProductPurchasedNotification object:productIdentifier
userInfo:nil];
}
```

One more thing …

Restoring completed transactions

FruitTableViewController.m

```
#pragma mark - User Interface set up
- (id)init {
    self = [super init];
    if (self) {
        self.title = @"Buy Fruits";
        self.navigationItem.rightBarButtonItem = [[UIBarButtonItem alloc]
initWithTitle:@"Restore" style:UIBarButtonItemStyleBordered target:self
action:@selector(restoreTapped:)];

        self.priceFormatter = [[NSNumberFormatter alloc] init];
        [self.priceFormatter setFormatterBehavior:NSNumberFormatterBehavior10_4];
        [self.priceFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];
    }
    return self;
}

...

#pragma mark - StoreKit
- (void)restoreTapped:(id)sender {
    [[FruitIAPHelper sharedInstance] restoreCompletedTransactions];
}
```

Agenda

- Motivation / Examples (in) applications
- Overview
- **Implementation**
  - iTunes Connect Metadata Setup
  - **StoreKit on iOS (+ exercise)**
    - StoreKit API overview
    - Getting product information
    - Purchasing
    - Restoring completed transactions
    - **Receipts**
  - Testing in Sandbox
- Summary

Receipts

- Are signed by Apple files
- Allow to securely verify the purchase of an In-App Purchase
- Were different on Mac OS / iOS... But, new with iOS 7: Same receipt format for iOS 7 and Mac OS X

- One receipt per purchase
- Receipt data made available by `SKTransaction`

Agenda

- Motivation / Examples (in) applications
- Overview
- **Implementation**
  - iTunes Connect Metadata Setup
  - **StoreKit on iOS (+ exercise)**
    - StoreKit API overview
    - Getting product information
    - Purchasing
    - Restoring completed transactions
    - Receipts
  - Testing in Sandbox
- Summary

In-App Purchase Tips

- Always call `finishTransaction` on all transactions! (most In-App Purchase errors) ...
- Register as `SKPaymentQueue` observer on launch! (in order to collect pending transactions)
- Only restore In-App Purchases when user needs access to In-App Purchases.

Agenda

- Motivation / Examples (in) applications
- Overview
  - What is an In-App Purchase?
  - Overview: types of In-App Purchases
- **Implementation**
  - iTunes Connect Metadata Setup
  - StoreKit on iOS (+ exercise)
  - **Testing in Sandbox**
- Summary

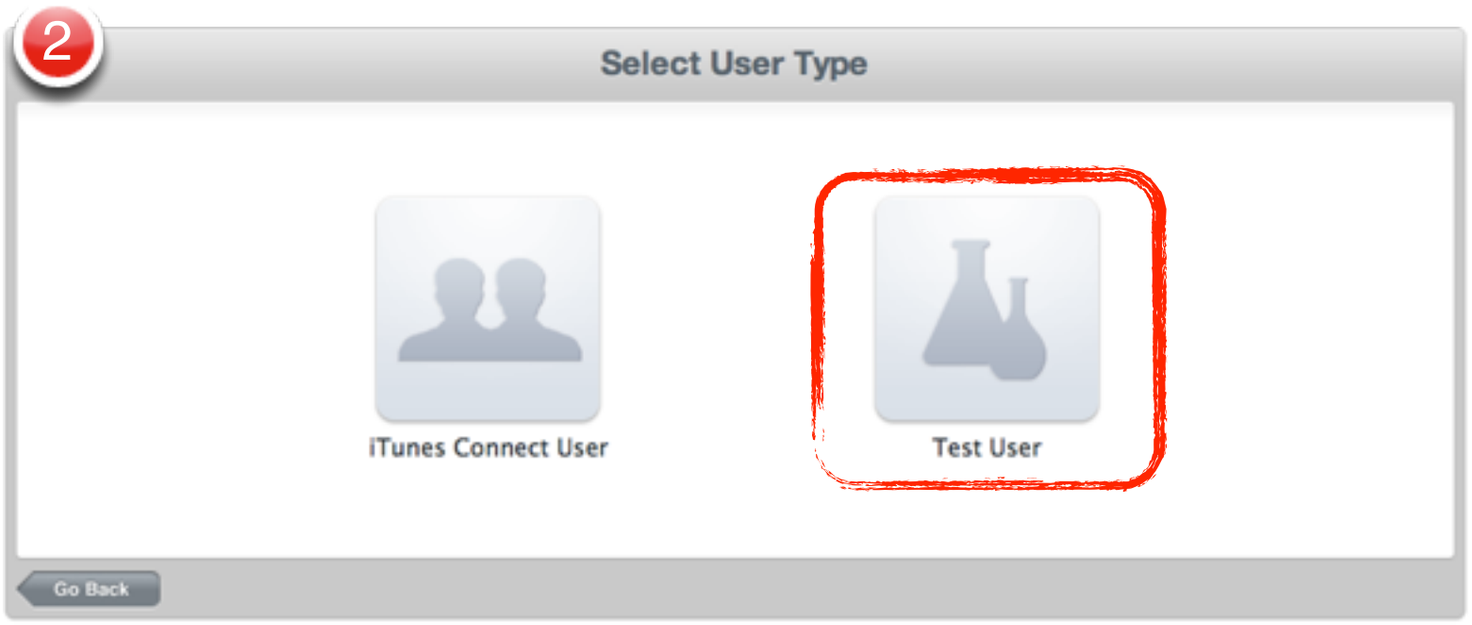
Testing in Sandbox

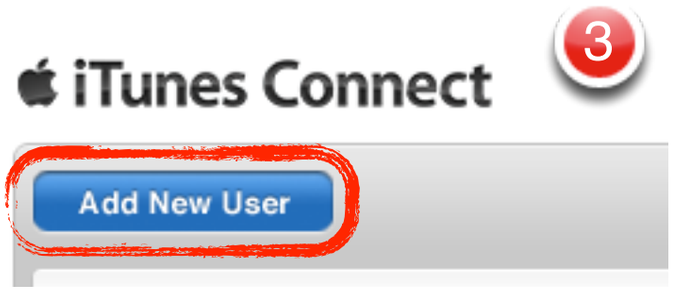
- Development-signed builds automatically use the sandbox environment.
- **Payments are not processed** in the sandbox environment, but **transactions are returned** = buying your own In-App products, without spending real money.
- You must create a **sandbox test user**, not a regular production store account for sandbox testing
- Sign out of iTunes / AppStore account first!

Testing in Sandbox

- Create a Test User in iTunes Connect ...

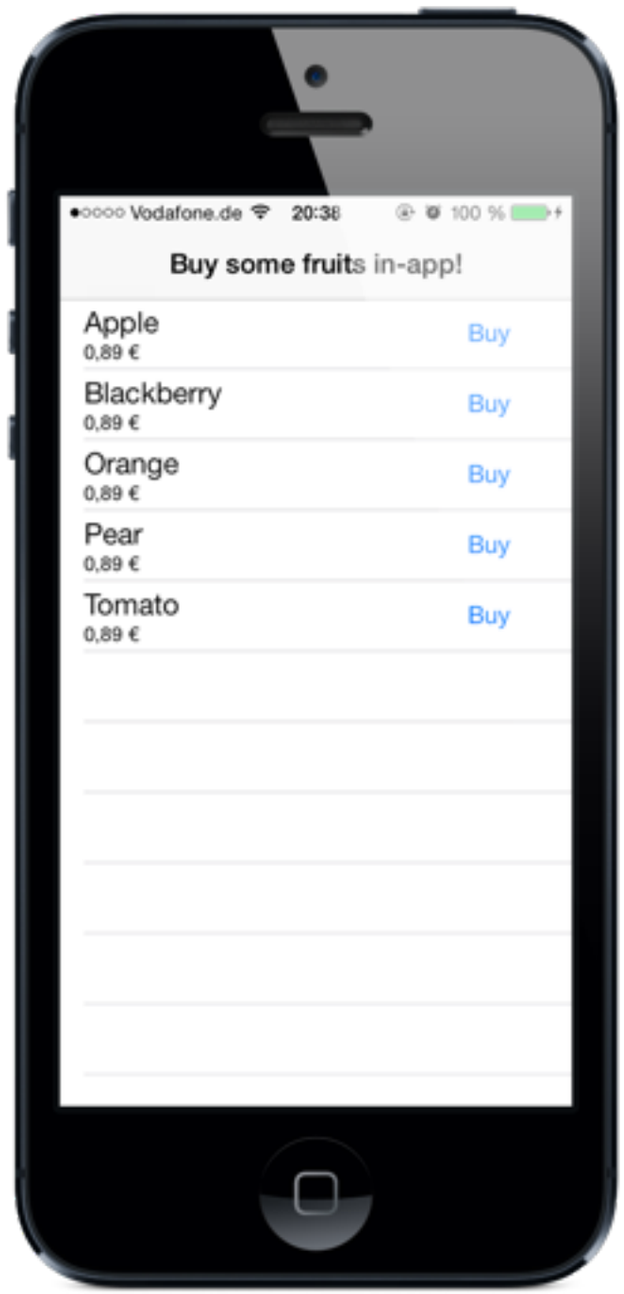
Manage Users
Add, view, and manage iTunes Connect users and In-App Purchase test accounts.





Testing in Sandbox

- Build & Run the App on a device...

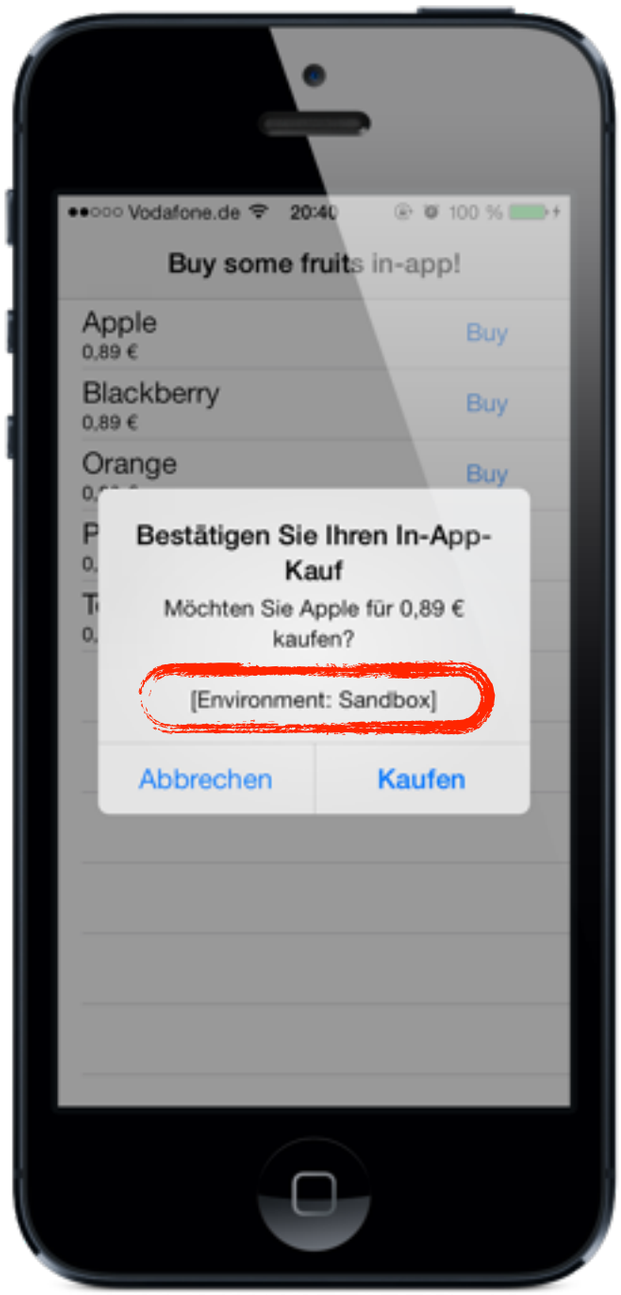


Testing in Sandbox

- You‘ll need to log in with the Test User Account we created...

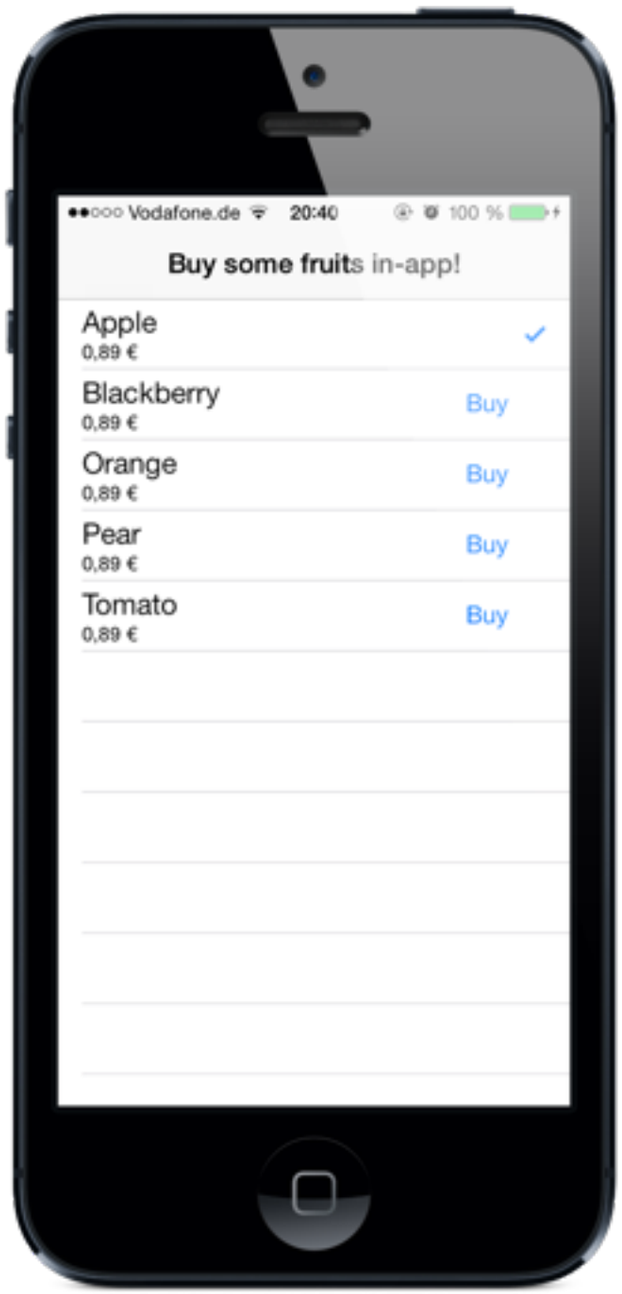
Testing in Sandbox

- Verify you‘re in the Sandbox environment & confirm the purchase



Testing in Sandbox

- Done!



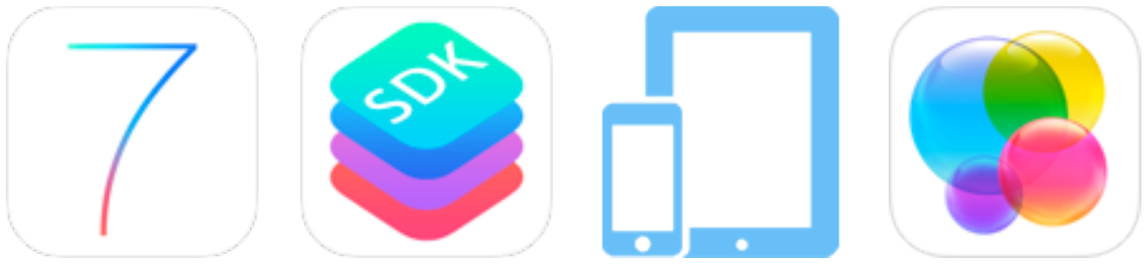
Agenda

- Motivation / Examples (in) applications
- Overview
  - What is an In-App Purchase?
  - Overview: types of In-App Purchases
- Implementation
  - iTunes Connect Metadata Setup
  - StoreKit on iOS (+ exercise)
  - Testing in Sandbox
- **Summary**

Summary

- 3 „real“ In-App Purchase types: **Non-consumable, Consumable, Auto-renewing subscription**.

Product type	Consumable	Non-consumable	Auto-renewable
Users can buy	Multiple times	Once	Multiple times
Synced across devices	Not synced	By the system	By the system
Restored	Not restored	By the system	By the system



Summary

- Auto-renewable subscriptions are complicated.
- **Restoration** of In-App Purchases is **mostly required**.
- **Use a device** for testing In-App Purchases.
- Have a Paid Applications Contract in effect with Apple.

Where To Go From Here?

- Hosted downloads from Apple‘s servers
- Fully Server-Based System: how to add new In-App purchases without updating your app
- `SKStoreProductViewController`: how to use the method to sell items from the iTunes Store in your app
- Download progress: show download progress and status

Outlook

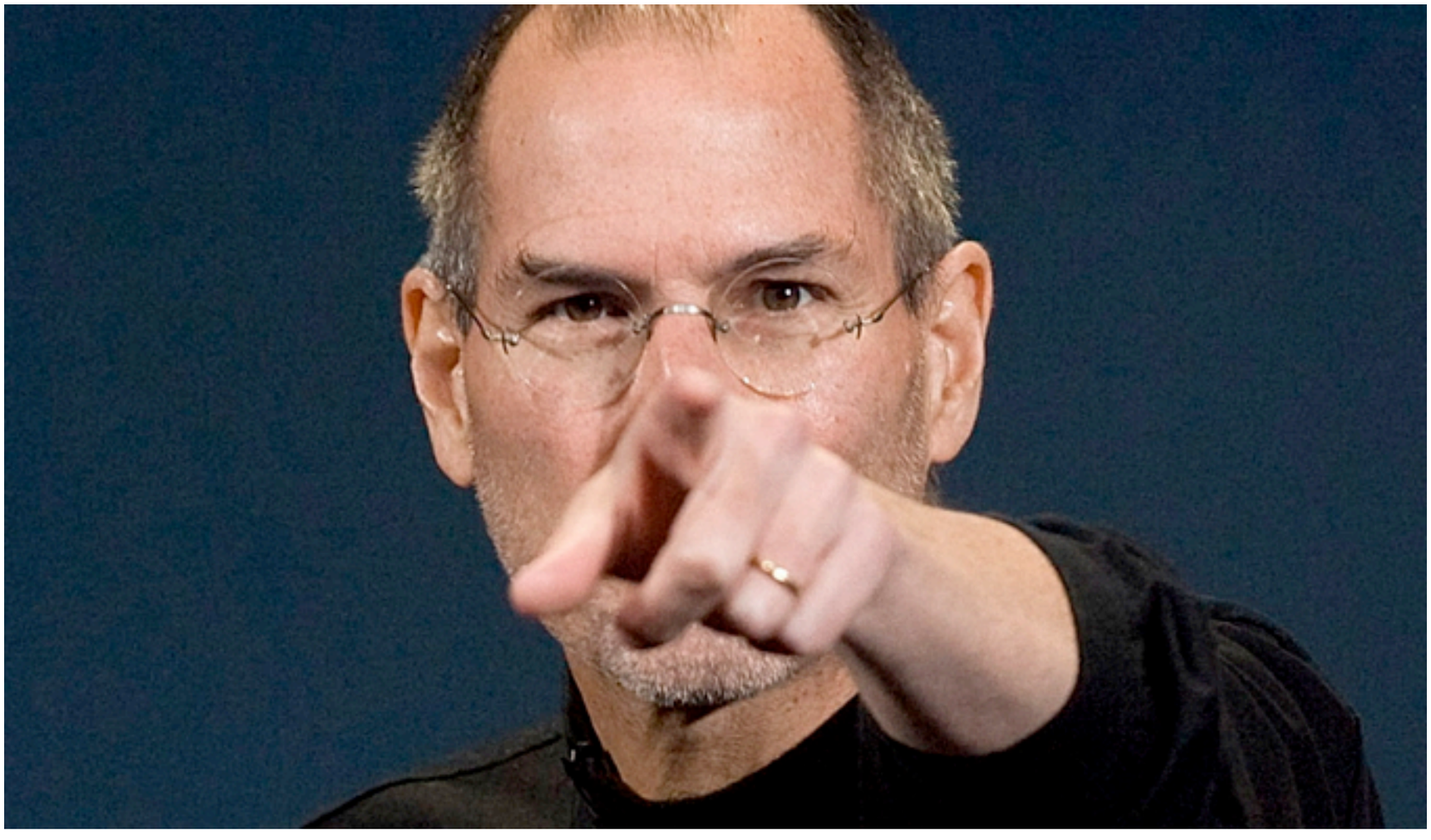
- The consulting firm PwC expects **substantial growth jumps** with the „free-to-play“ business model.
- Players in Germany have spent in 2012 **260 million € for virtual goods**. By 2016, there will be **over 400 million €** spent.
- Almost half of all video and computer gamers is now **willing to pay for digital assistants or a higher level of play**.
- "The **high growth rates** also encourage other game companies to rethink their strategies and **focus on virtual goods**" – PwC's technology expert Werner Ballhaus.

Source: Handelsblatt, August 22, 2013, Nr. 161 and pwc.com

Q&A

Thank you for your attention!

Please feel free to ask questions! :-)



References

- https://developer.apple.com/in-app-purchase/
- https://developer.apple.com/in-app-purchase/In-App-Purchase-Guidelines.pdf
- https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Introduction/Introduction.html
- https://developer.apple.com/library/ios/documentation/StoreKit/Reference/StoreKit_Collection/_index.html
- https://developer.apple.com/library/ios/technotes/tn2259/_index.html

References

- https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/1_Introduction/Introduction.html
- https://developer.apple.com/library/ios/technotes/tn2259/_index.html#//apple_ref/doc/uid/DTS40009578-CH1-FREQUENTLY_ASKED_QUESTIONS
- https://developer.apple.com/library/ios/qa/qa1329/_index.html
- https://developer.apple.com/appstore/guidelines.html
- http://www.raywenderlich.com/21081/introduction-to-in-app-purchases-in-ios-6-tutorial