creating App on iTunes Connect

add new app

Consumable → Refference name: One Meal; Product ID: com.iapcourse.meal

Non Consumable → Display Name: meal & lower letter

Auto-Reneable Subscription

Non-Renewing subscription

// sandbox testers

↓ Users and Roles ⇒ add Sandbox Tester

* hold shift and select all in storyboard to put in stack view

// Product.swift (Enums)

* Do not verify the email for the sandbox account

```
import Foundation

enum Product: Int{
    case hideAds = 0
    case meal = 1
}
```

// this will hide in-app advertising

// PurchaseStatus.swift (Enums)

```
import Foundation

enum PurchaseStatus{
    case purchased
    case restored
    case failed

}
```

// FoodItems (Helpers)

```
import Foundation
let defaultPrice = 9.99
let salmon = Item(image: UIImage(named: "food1")!, name: "Salmon", price: defaultPrice)
```

↓

```
let foodItems: [Items] = [salm, ..... ]
```

// Constants.swift (Helpers)

```
import Foundation

// MARK: IAP Identifiers
let IAP_MEAL_ID = "com.iapcourse.meal"
let IAP_HIDE_ADS_ID = "com.iapcourse.hideads"

// MARK: Notification Identifiers
let IAPServicePurchaseNotification = "IAPServicePurchaseNotification"
let IAPServiceRestoreNotification = "IAPServiceRestoreNotification"
let IAPServiceFailureNotification = "IAPServiceFailureNotification"
```

```
// IAPService.swift  (services)
import Foundation
import StoreKit

protocol IAPServiceDelegate{
    func iapProductsLoaded()
}

class IAPService : NSObject, SKProductsRequestDelegate{
    static let instance = IAPService()
    var delegate : IAPServiceDelegate?

    var products = [SKProduct]()
    var productIds = Set<String>()
    var productRequest = SKProductsRequest()

    var nonConsumablePurchaseWasMade = UserDefaults.standard.bool(forKey: "nonConsumablePurchaseWasMade")

    override init(){
        super.init()                          ← !add this class as observer!
        SKPaymentQueue.default().add(self)
    }
    func loadProducts(){
        productId ToStringSet()
        requestProducts(forIds: productIds)
    }
    func productIdToStringSet(){
        let ids = [IAP_HIDE_ADS_ID, IAP_MEAL_ID]      ✗ reversed order ⇒ try directly convert to set in real app
        for id in ids{
            productIds.insert(id)
        }
    }

    func requestProducts(forIds ids: Set<String>){
        productRequest.cance(
        productRequest = SKProductRequest(productIdentifiers: ids)
        productRequest.delegate = self
        productRequest.start()
    }
                        ↗ come with protocol
    func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse){
        self.products = response.products

        if products.count == 0{
            requestProducts(forIds: productIds)
        }else{
            delegate?.iapProductsLoaded()
            //print(products[0].localizedTitle)
        }
    }
```

```
func attemptPurchaseForItemWith(productIndex: Product){
    let product = products[productIndex.rawValue]
    let payment = SKPayment(product: product)
    SKPaymentQueue.default().add(payment)
}

func restorePurchases(){
    SKPaymentQueue.default().restoreCompletedTransactions()
}

}
```

```
extension IAPService: SKPaymentTransactionObserver {
    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                SKPaymentQueue.default().finishTransaction(transaction)
                complete(transaction: transaction)
                sendNotificationFor(status: .purchased, withIdentifier: transaction.payment.productIdentifier)
                debugPrint("Purchase was successful!")
                break
            case .restored:
                SKPaymentQueue.default().finishTransaction(transaction)
                break
            case .failed:
                SKPaymentQueue.default().finishTransaction(transaction)
                sendNotificationFor(status: .failed, withIdentifier: nil)
                break
            case .deferred:
                break
            case .purchasing:
                break
            }
        }
    }

    func paymentQueueRestoreCompletedTransactionsFinished(_ queue: SKPaymentQueue) {
        sendNotificationFor(status: .restored, withIdentifier: nil)
        setNonConsumablePurchase(true)
    }

    func complete(transaction: SKPaymentTransaction) {
        switch transaction.payment.productIdentifier {
        case IAP_MEAL_ID:
            break
        case IAP_HIDE_ADS_ID:
            setNonConsumablePurchase(true)
            break
        default:
            break
        }
    }

    func setNonConsumablePurchase(_ status: Bool) {
        UserDefaults.standard.set(status, forKey: "nonConsumablePurchaseWasMade")
    }

    func sendNotificationFor(status: PurchaseStatus, withIdentifier identifier: String?) {
        switch status {
        case .purchased:
            NotificationCenter.default.post(name: NSNotification.Name(IAPServicePurchaseNotification), object: identifier)
            break
        case .restored:
            NotificationCenter.default.post(name: NSNotification.Name(IAPServiceRestoreNotification), object: nil)
            break
        case .failed:
            NotificationCenter.default.post(name: NSNotification.Name(IAPServiceFailureNotification), object: nil)
            break
        }
    }
}
```

← specific for restore

// Item.swift (Model)

```
import UIKit


class Item{
    public private(set) var image
    public private(set) var name
    public private(set) var price

    init ( image : UIImage , name : String , price : Double){
        self. image = image
        self. name = name
        self . price = price
    }
}
```

// ItemCell.swift (view)

```
import UIKit


class ItemCell : UICollectionViewCell{
    @IBOutlet weak var itemImageView : UIImageView!
    @IBOutlet weak var itemNameLbl : UILabel!
    @IBOutlet weak var itemPriceLbl : UILabel!

    func configureCell (forItem item : Item){
        itemImageView. image = item. Image
        itemName Lbl . text = item, name
        Item Price Lbl . text = String( describing : Item, price)
    }
```

// StorefrontVC.swift (Controller)

```
import UIKit

class StorefrontVC : UIViewController, UICollectionViewDelegate, UICollectionViewDataSource{

    @IBOutlet weak var collectionView : UICollectionView!

    override func viewDidLoad (){
        super. viewDidLoad ()
        collection View . delegate = self
        collection View . dataSource = self
        IAPService . instance . delegate = self
        IAPService . instance . load Products ()

        Notification Center . default . addObserver ( self , selector : #selector ( showRestored Alert )), name :
            NSNotification. Name (IAPService Restore Notification), object : nil)
    }
```

```
func numberOfSections(in collectionView: UICollectionView) -> Int {
    return 1
}

func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    return foodItems.count
}

func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
    guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "itemCell", for: indexPath)
        as? ItemCell else { return UICollectionViewCell() }
    let item = foodItems[indexPath.row]
    cell.configureCell(forItem: item)
    return cell
}

func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {
    guard let detailVC = storyboard?.instantiateViewController(withIdentifier: "DetailVC") as? DetailVC else {
        return }
    let item = foodItems[indexPath.row]
    detailVC.initData(forItem: item)
    presentDetail(detailVC, animated: true, completion: nil)
}
```

```
    @objc func showRestoredAlert() {
        let alert = UIAlertController(title: "Success!", message: "Your purchases were successfully restored.", preferredStyle: .alert)
        let action = UIAlertAction(title: "OK", style: .default, handler: nil)
        alert.addAction(action)
        present(alert, animated: true, completion: nil)
    }

    @IBAction func restoreBtnWasPressed(_ sender: Any) {
        let alert = UIAlertController(title: "Restore Purchases?", message: "Do you want to restore any in-app purchases you've previously purchased?", preferredStyle: .actionSheet)
        let action = UIAlertAction(title: "Restore", style: .default) { (action) in
            IAPService.instance.restorePurchases()
        }
        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
        alert.addAction(action)
        alert.addAction(cancelAction)
        present(alert, animated: true, completion: nil)
    }
}
```

```
extension StorefrontVC: IAPServiceDelegate {
    func iapProductsLoaded() {
        print("IAP products loaded")
    }
}
```

```
//Detail VC. swift (controller)
import UIKit                                    // set the stack view to fill alignment

class DetailVC: UIViewController {
    @IBOutlet weak var itemImageView: UIImageView!
    @IBOutlet weak var itemNameLabel: UILabel!
    @IBOutlet weak var itemPriceLabel: UILabel!
    @IBOutlet weak var uglyAdView: UIView!
    @IBOutlet weak var buyItemBtn: UIButton!
    @IBOutlet weak var hideAdsBtn: UIButton!

    public private(set) var item: Item!
    private var hiddenStatus: Bool = UserDefaults.standard.bool(forKey: "nonConsumblePurchaseWasMade")

    func initData(forItem item: Item){
        self.item = item
    }

    override func viewDidLoad(){
        super.viewDidLoad()

        itemImageView.image = item.image
        itemNameLbl.text = item.name
        itemPriceLbl.text = String(describing: item.price)
        buyItemBtn.setTitle("Buy this item for $\(item.price)", for: .normal)

        NotificationCenter.default.addObserver(self, selector: #selector(handlePurchase(_:)), name:
            NSNotification.Name(IAPServicePurchaseNotification), object: nil)
        NotificationCenter.default.addObserver(self, selector: #selector(handleFailure(_:)), name:
            NSNotification.Name(IAPServiceFailureNotification), object: nil)

    }

    override func viewWillAppear(_ animated: Bool){
        super.viewWillAppear(animated)
        showOrHideAds()
    }

    override func viewWillDisappear(_ animated: Bool){
        super.viewWillDisappear(animated)
        NotificationCenter.default.removeObserver(self)
    }
    @objc handlePurchase(_ notification: Notification){
        guard let productID = notification.object as? String else {return}
```

```
        switch productID {
        case IAP_MEAL_ID:
            buyItemBtn.isEnabled = true
            break
        case IAP_HIDE_ADS_ID:
            uglyAdView.isHidden = true
            hideAdsBtn.isHidden = true
            break
        default:
            break
    }

    @objc func handleFailure(){
        buyItemBtn.isEnabled = true


    }

    func showOrHideAds(){
        uglyAdView.isHidden = hiddenStatus
        hideAdsBtn.isHidden = hiddenStatus
    }


    @IBAction func buyBtnWasPressed(_ sender: Any){
        buyItemBtn.isEnabled = false
        IAPService.instance.attemptPurchaseForItemWith(productIndex: .meal)
    }

    @IBAction func hideAdsBtnWasPressed(_ sender: Any){
        IAPService.instance.attemptPurchaseForItemWith(productIndex: .hideAds)

    }



@IBAction func closeBtnWasPressed(_ sender: Any){
    dismiss(animated: true, completion: nil)
}
```