

iOS授業8日目資料

課題発表

はじめに

はじめに：今日以降の授業

日付	タイトル	内容
11/2	Firebase Storage	・ Firebase Storageに画像を保存する
11/9	Cloud Functions	・ サーバー側から処理を実施する

はじめに：今日やること

◆1.CloudStorageについて

◆2.CloudStorageセ ッ ト ア ッ プ

◆3.CloudStorageで保存・読み込みをする

◆4.(時間あれば)削除とか読み込みくるくるとか

Cloud Storageについて & セットアップ

Cloud Storageについて

- Firebaseのストレージサービス
- Firestoreに保存できない、写真・動画・音声などのデータファイルをクラウド上に保存するときに使用する
- あの有名なSpotifyもこのクラウドストレージを使用している
- 全員に見せるべきでないファイルの場合は
Firestoreのようにセキュリティルールを設定する

<https://firebase.google.com/docs/storage/?hl=ja>

準備

本日配布している

「GsTodo」のプロジェクトを
手元に準備してください

GoogleService-Info.plistと
自分のものに置き換えてください

【一緒にやってみよう】

Cloud Storageのセットアップをしよう

Cloud Storageセットアップ

コンソールでStorageを選択して「スタートガイド」を選択



Cloud Storage セットアップ

ポップアップが出るので、内容を確認してそのまま進む
(これやらないとストレージが生成されない)



補足：ToDoアプリ用にStorageのルールを設定してみる

```
service firebase.storage {  
  match /b/{bucket}/o {  
  
    allow read, write: if false;  
  
    function isAuthenticated() {  
      return request.auth != null;  
    }  
  
    match /{userId}/{allPaths=**} {  
  
      function isUserAuthenticated() {  
        return request.auth.uid == userId;  
      }  
  
      allow read, write: if isAuthenticated() && isUserAuthenticated();  
    }  
  }  
}
```

全て禁止！

ユーザーが認証されているか
チェックする関数

Refを指定

上のuserIdとリクエスト元の
ユーザIDが同一か
チェックする関数

関数が両方trueだったら
読み込みと書き込みを
許可している

ToDoアプリ用にFirestoreのルールを設定してみる

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

—— ユーザーが認証されているか
チェックする

これが最も簡単なセキュリティ。最低限は設定する。
複雑なアプリになるときちんと設定しなければいけない。

Cloud Storageセ ッ ト ア ッ プ

いつものpod install

PodFileに追加して…

```
pod 'Firebase/Storage'
```

Podコマンド

```
$ pod install
```

ここまでのおさらい

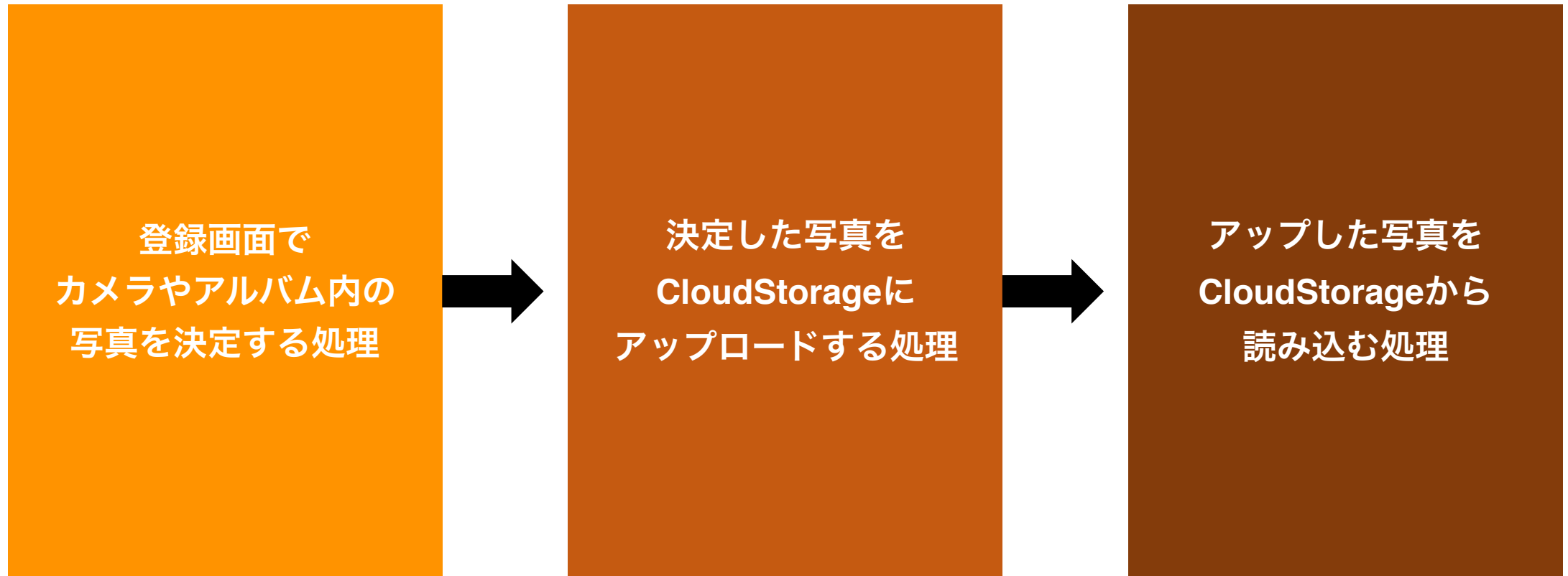
こんなことを学びました

①CloudStorageについて

②CloudStorageのセットアップ方法

Cloud Storageでのアップロード

画像周りの処理を大きく3つの工程に分けます

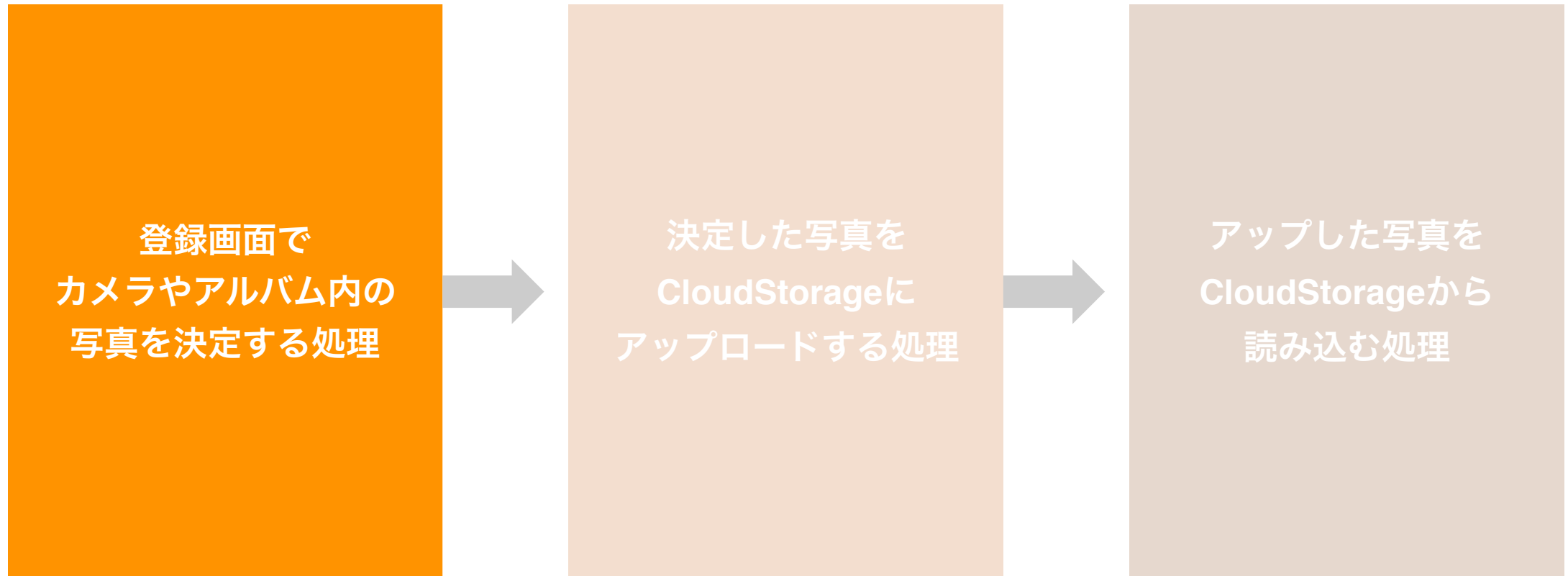


【一緒にやってみよう】

カメラやアルバムで写真を選択する処理を作ろう

※完成PJは配布するので
ついてこれなくなったら
手を止めて見ることに集中！

ここやります

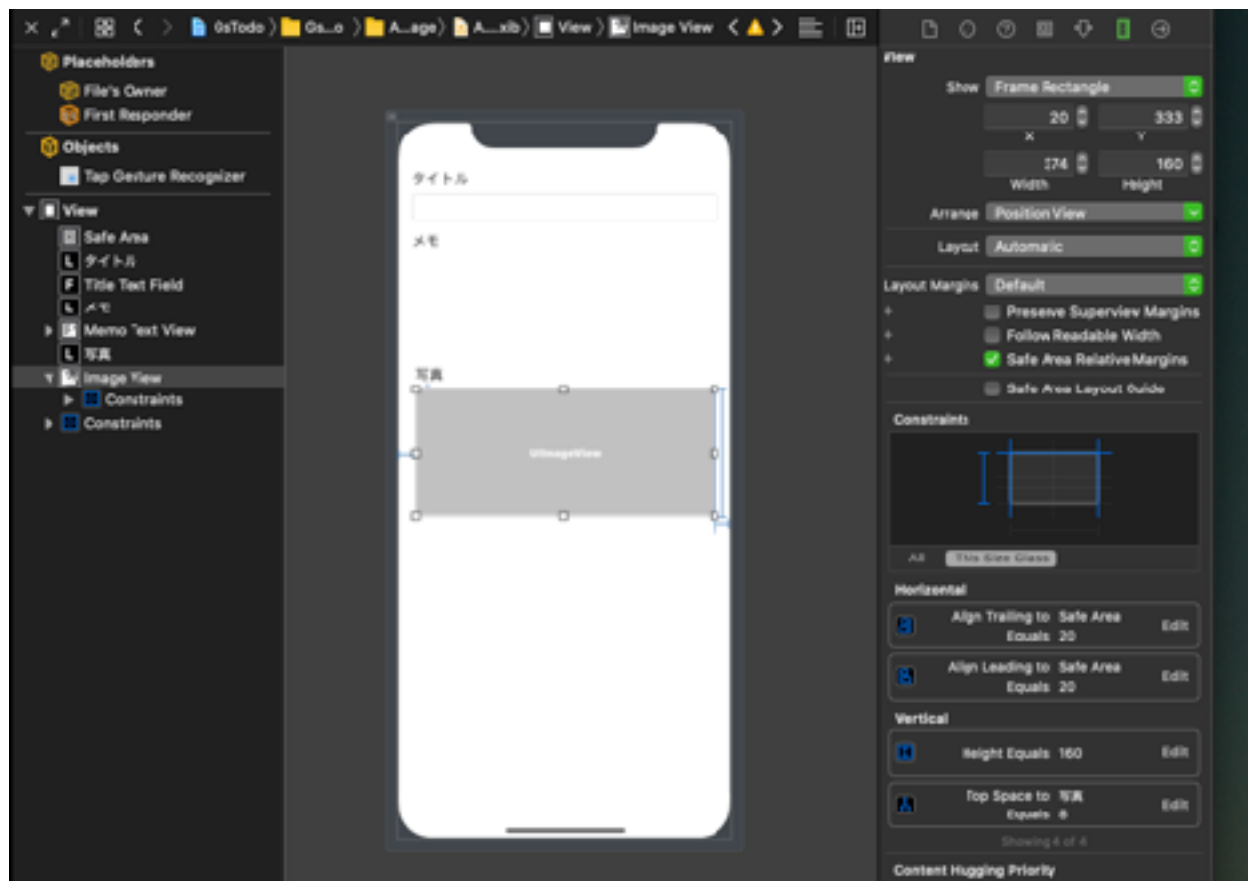


やること（～写真決定まで）

- ①AddVCの画面に写真欄を追加
- ②写真欄タップで「アルバムor撮影」を選択させる
- ③各種Privacy系の許可をplistで設定
- ④「アルバム」と「撮影」で写真を決定する処理を作る

①AddVCの画面に写真欄を追加

AddViewController



1-1. 写真欄を追加して
IBOutletで「imageView」を追加

1-2. 写真欄に
GestureRecognizerを追加して
タップイベントをIBActionで拾う

1-3. 写真欄のUser Interactionを
有効化する(超重要！)

②写真欄タップで「アルバムor撮影」を選択させる

AddViewController

1-2で作った処理で、アクションシートを使って、選択肢を出す

```
@IBAction func tapImageView(_ sender: Any) {
    print("🌞 imageView をタップしたよ")
    // アクションシートを表示する

    let alertSheet = UIAlertController(title: nil, message: "選択してください", preferredStyle: .actionSheet)
    let cameraAction = UIAlertAction(title: "カメラで撮影", style: .default) { action in
        print("カメラが選択されました")
    }

    let albumAction = UIAlertAction(title: "アルバムから選択", style: .default) { action in
        print("アルバムが選択されました")
    }

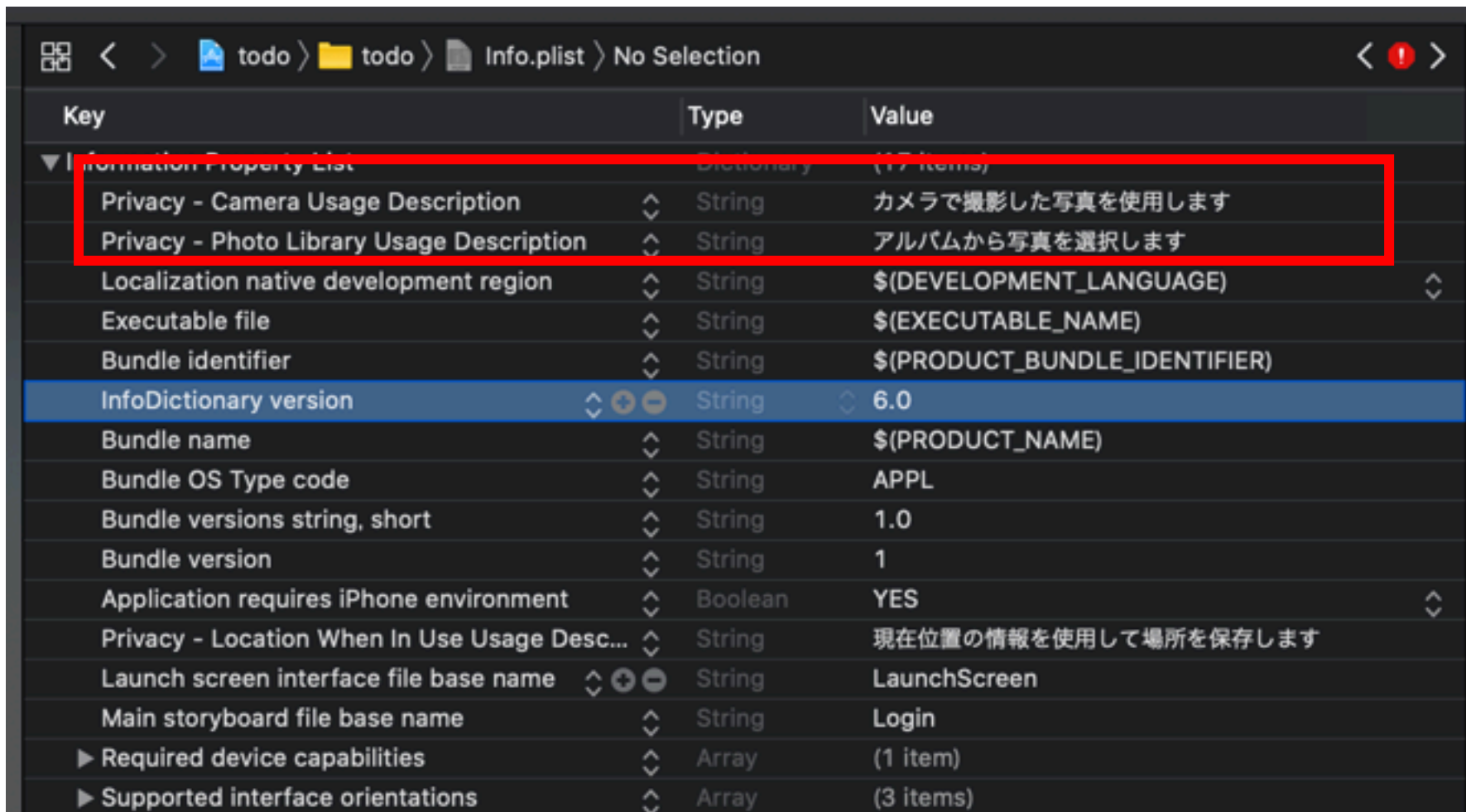
    let cancelAction = UIAlertAction(title: "キャンセル", style: .cancel) { action in
        print("キャンセルが選択されました")
    }

    alertSheet.addAction(cameraAction)
    alertSheet.addAction(albumAction)
    alertSheet.addAction(cancelAction)

    present(alertSheet, animated: true)
}
```

③各種Privacy系の許可をplistで設定 Info.plist

カメラとフォトライブラリのプライバシー説明文を追加



The screenshot shows the Xcode interface for editing an Info.plist file. The breadcrumb path at the top is 'todo > todo > Info.plist > No Selection'. The table below lists various keys and their values. Two keys are highlighted with a red box: 'Privacy - Camera Usage Description' and 'Privacy - Photo Library Usage Description'.

Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
Privacy - Camera Usage Description	String	カメラで撮影した写真を使用します
Privacy - Photo Library Usage Description	String	アルバムから写真を選択します
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Privacy - Location When In Use Usage Desc...	String	現在位置の情報を使用して場所を保存します
Launch screen interface file base name	String	LaunchScreen
Main storyboard file base name	String	Login
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)

④ 「アルバム」と「撮影」で写真を決定する処理を作る

AddViewController

4-1. アルバムとカメラの画面を生成する関数を作る

```
extension AddViewController: UINavigationControllerDelegate,
UIImagePickerControllerDelegate {

    func presentPicker (sourceType: UIImagePickerController.SourceType) {
        if UIImagePickerController.isSourceTypeAvailable(sourceType) {
            let picker = UIImagePickerController()
            picker.sourceType = sourceType
            picker.delegate = self
            present(picker, animated: true, completion: nil)
        } else {
            print ("The SouceType is not found.")
        }
    }
}
```


④ 「アルバム」と「撮影」で写真を決定する処理を作る

AddViewController

4-2. タップイベントの関数の中に以下の赤字部分を追記する

```
@IBAction func tapImageView(_ sender: Any) {
    print("🌞 imageView をタップしたよ")
    // アクションシートを表示する

    let alertSheet = UIAlertController(title: nil, message: "選択してください", preferredStyle: .actionSheet)
    let cameraAction = UIAlertAction(title: "カメラで撮影", style: .default) { action in
        print("カメラが選択されました")
        self.presentPicker(sourceType: .camera)
    }
    let albumAction = UIAlertAction(title: "アルバムから選択", style: .default) { action in
        print("アルバムが選択されました")
        self.presentPicker(sourceType: .photoLibrary)
    }
    let cancelAction = UIAlertAction(title: "キャンセル", style: .cancel) { action in
        print("キャンセルが選択されました")
    }
    alertSheet.addAction(cameraAction)
    alertSheet.addAction(albumAction)
    alertSheet.addAction(cancelAction)

    present(alertSheet, animated: true)
}
```

④ 「アルバム」と「撮影」で写真を決定する処理を作る

AddViewController

4-3. 撮影したり写真選択したときの処理を作る

```
// 撮影もしくは画像を選択したら呼ばれる
func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any]) {
    print("撮影もしくは画像を選択したよ!")

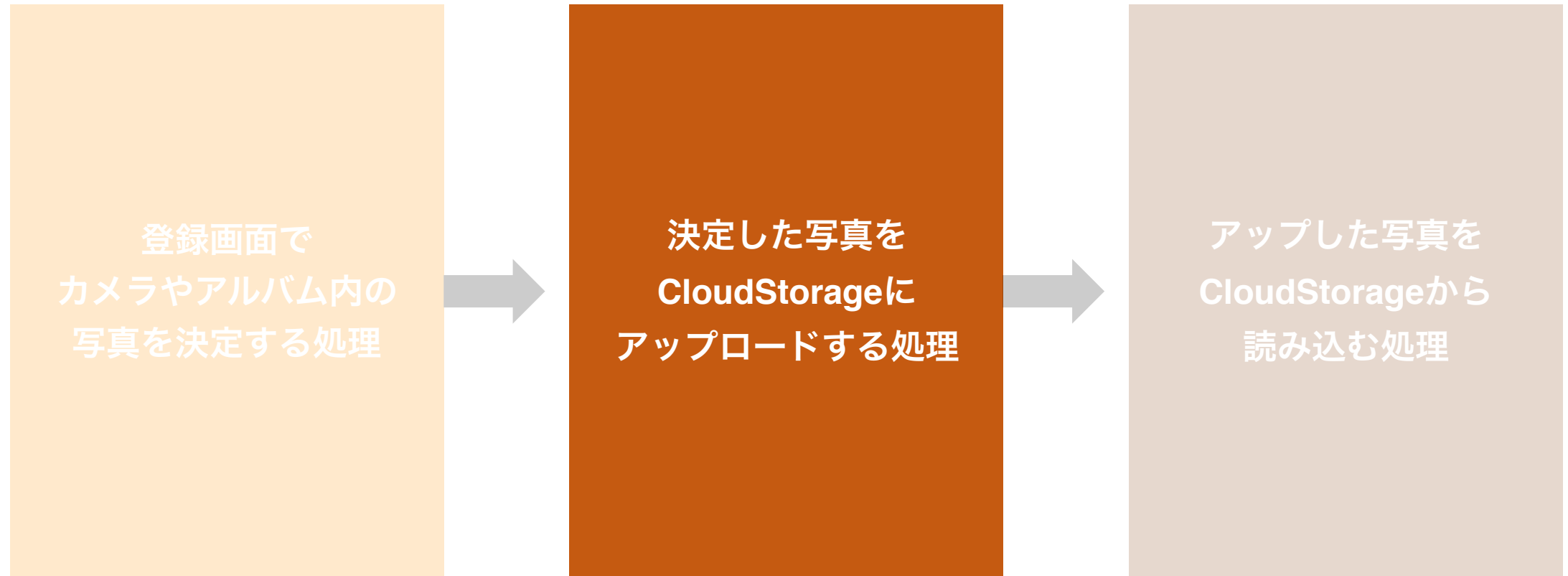
    if let pickedImage = info[.originalImage] as? UIImage {
        imageView.contentMode = .scaleAspectFit
    }
    // 表示した画面を閉じる処理
    picker.dismiss(animated: true, completion: nil)
}
```

【一緒にやってみよう】

写真をアップロードする処理を作ろう

※完成PJは配布するので
ついてこれなくなったら
手を止めて見ることに集中！

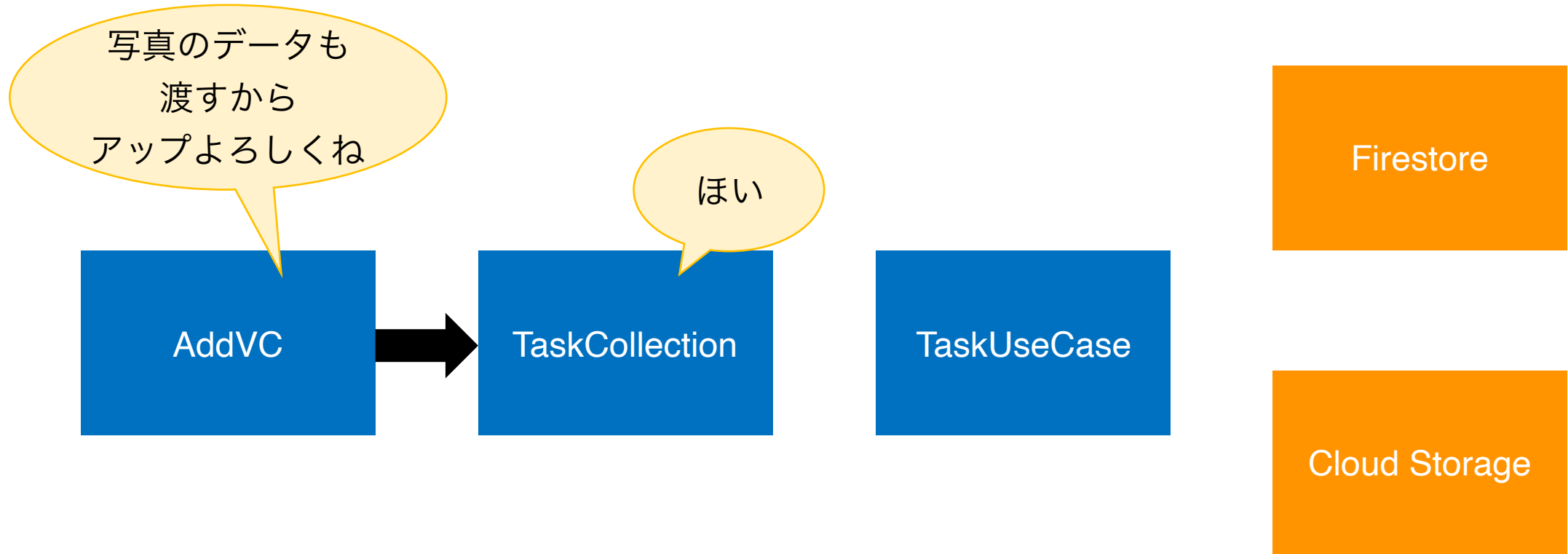
ここやります



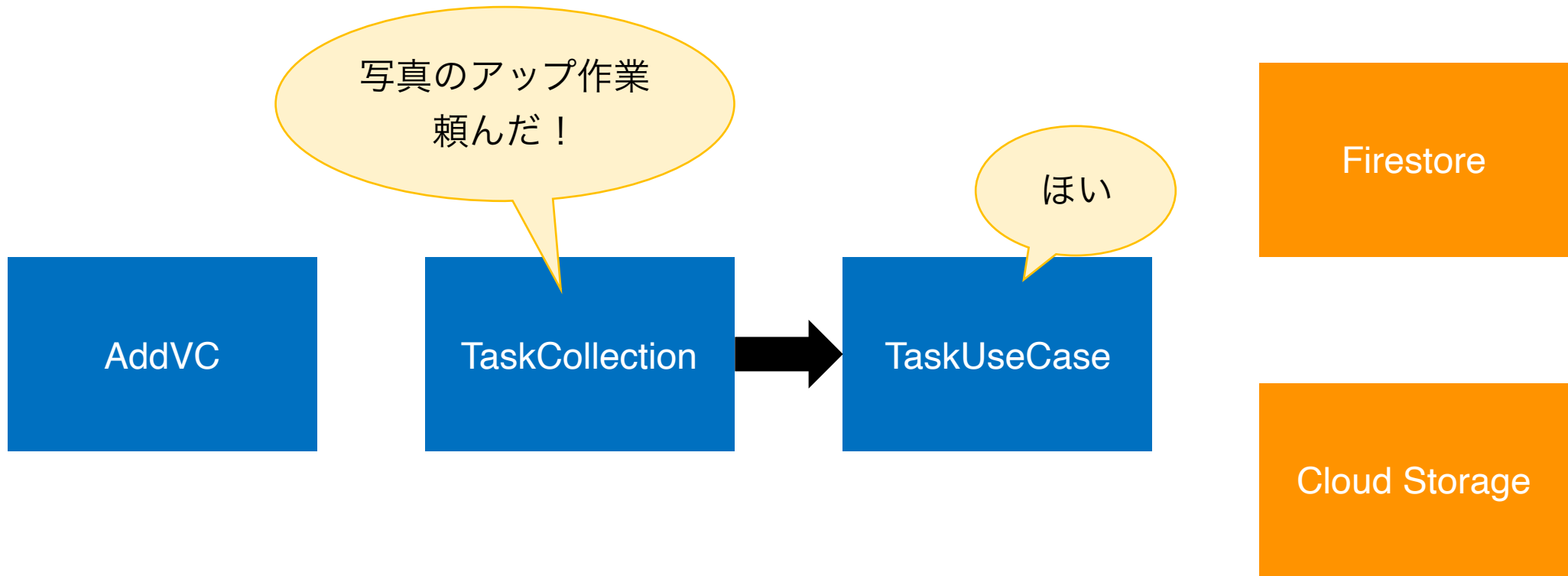
やること（～アップロードまで）

- ①写真をリサイズする処理を作る
- ②Taskに画像名を保存できるようにする
- ③写真をCloudStorageにアップする処理を作る
- ④アップした場所をToDoにセットする

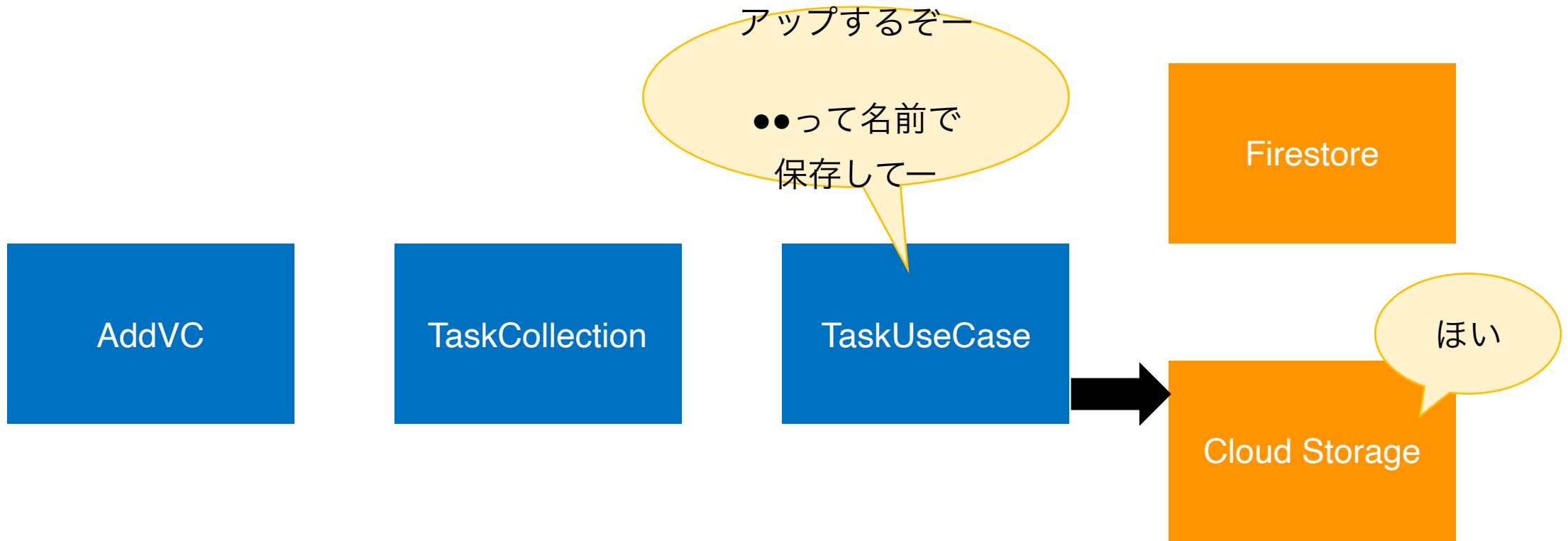
着手する前に今回の処理のチェック



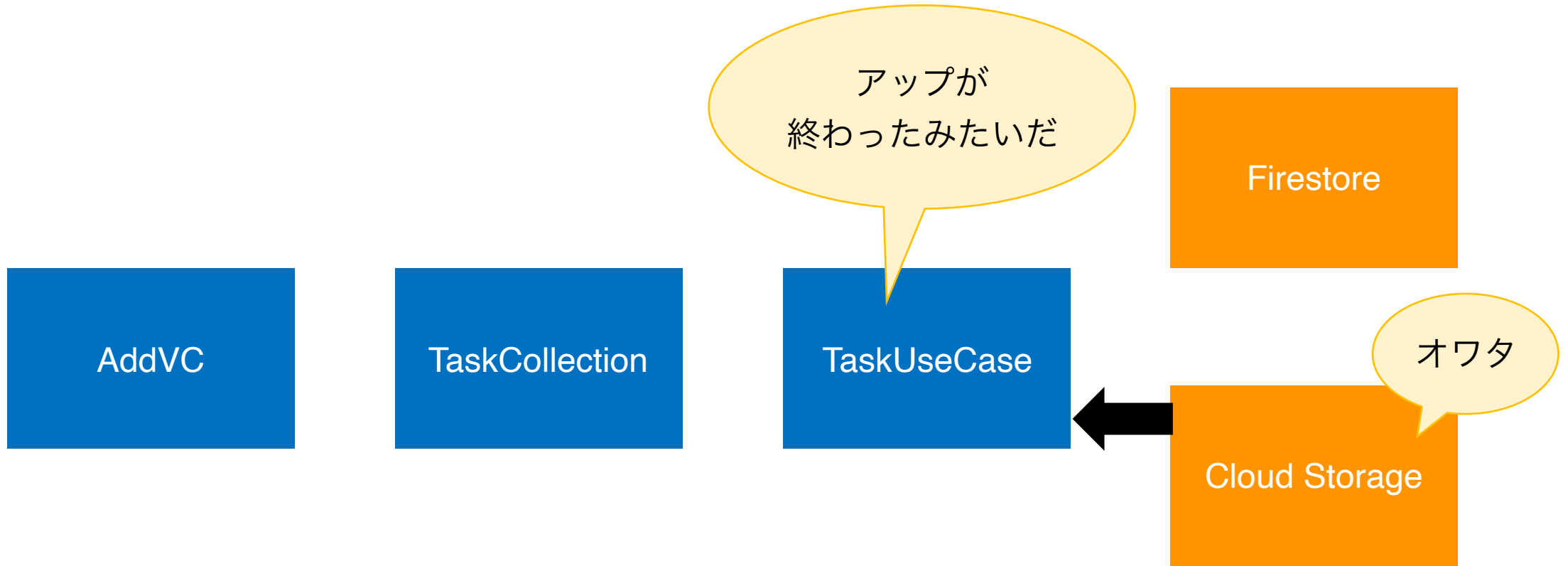
着手する前に今回の処理のチェック



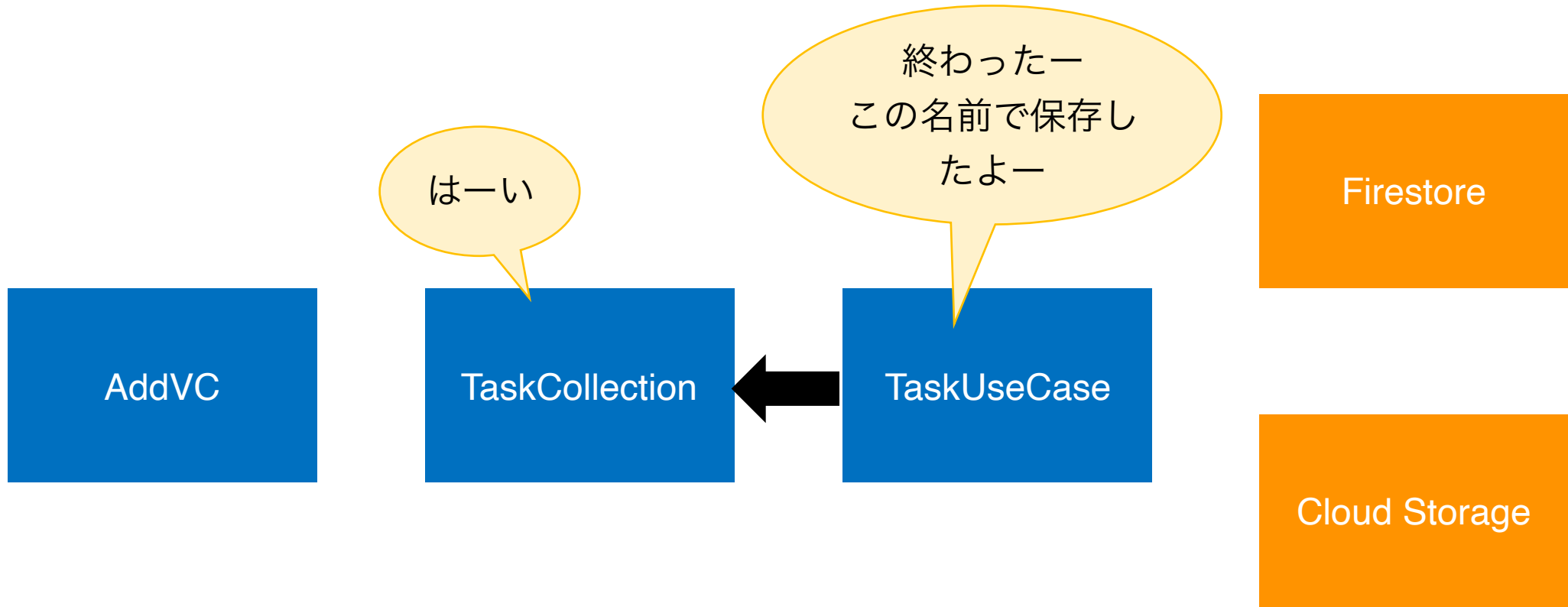
着手する前に今回の処理のチェック



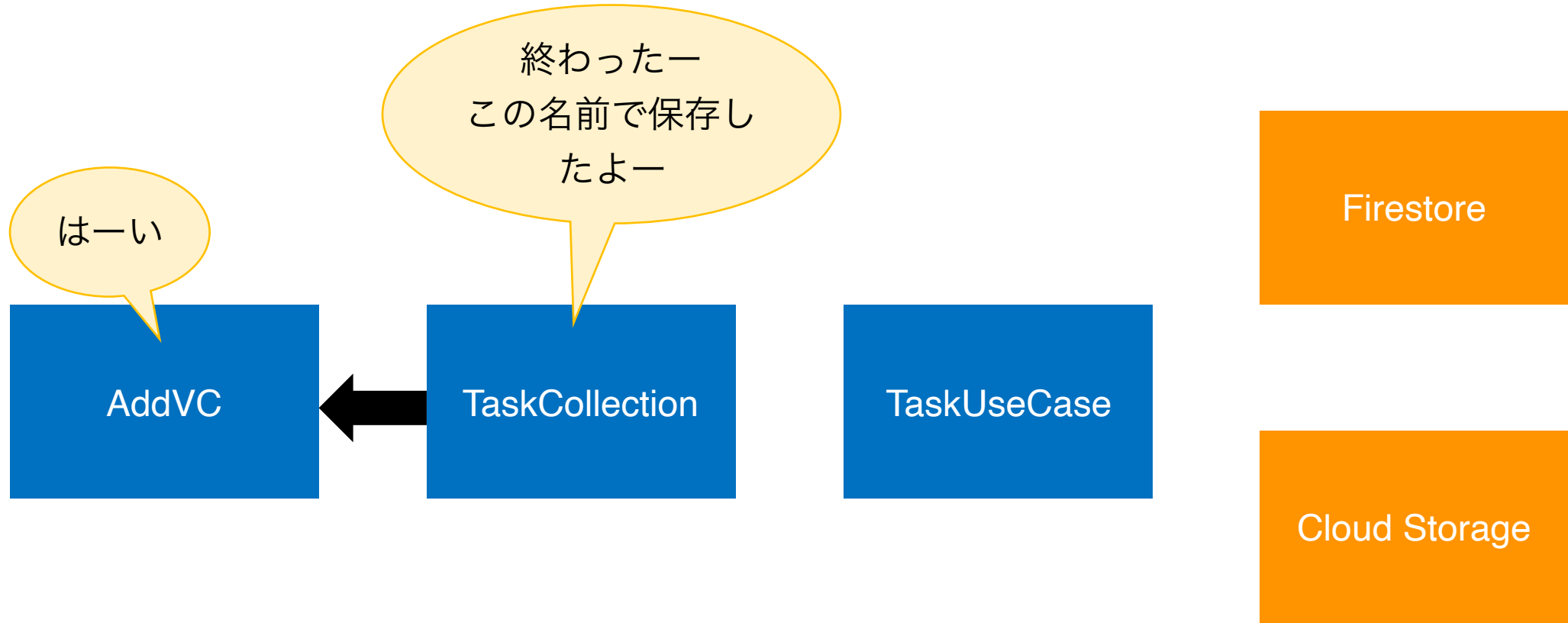
着手する前に今回の処理のチェック



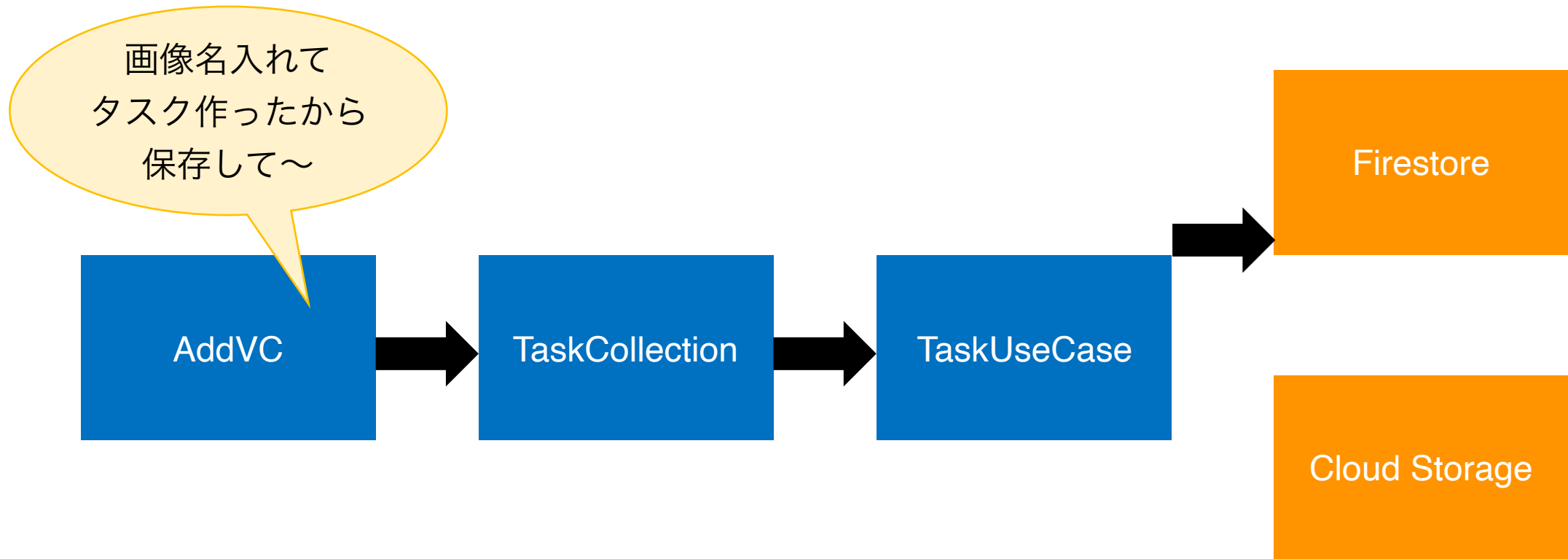
着手する前に今回の処理のチェック



着手する前に今回の処理のチェック



着手する前に今回の処理のチェック



①写真をリサイズする処理を作る UIImage+Resize

1-1. 横幅を指定してリサイズできるExtensionを作成する

```
import UIKit

extension UIImage {
    func resize(toWidth width: CGFloat) -> UIImage? {
        let canvasSize = CGSize(width: width,
                                height: CGFloat(ceil(width/size.width * size.height)))
        UIGraphicsBeginImageContextWithOptions(canvasSize, false, scale)
        defer { UIGraphicsEndImageContext() }
        draw(in: CGRect(origin: .zero, size: canvasSize))
        return UIGraphicsGetImageFromCurrentImageContext()
    }
}
```

①写真をリサイズする処理を作る AddViewController

1-2. リサイズ後の画像をセットするように修正

```
// 撮影もしくは画像を選択したら呼ばれる
func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any]) {
    print("撮影もしくは画像を選択したよ!")

    if let pickedImage = info[.originalImage] as? UIImage {
        imageView.contentMode = .scaleAspectFit
        imageView.image = pickedImage.resize(toWidth: 300)
    }
    // 表示した画面を閉じる処理
    picker.dismiss(animated: true, completion: nil)
}
```

② Taskに画像名を保存できるようにする

Task

```
import Foundation
import FirebaseFirestore
import FirebaseFirestoreSwift
// Task のクラス。
// プロパティに title と memo を持っている
class Task: Codable {
    var id: String
    var title: String = ""
    var memo: String = ""
    var imageName: String?
    var createdAt: Timestamp
    var updatedAt: Timestamp

    // init とは、Task を作るときに呼ばれるメソッド。(イニシャライザという)
    // 使い方: let task = Task(title: "プログラミング")
    init(id: String) {
        self.id = id
        self.createdAt = Timestamp() // Timestamp は、Firestore で使用できる便利な時間の方
        self.updatedAt = Timestamp()
    }
}
```

③写真をCloudStorageにアップする処理を作る

3-1. TaskUseCaseにアップロード処理を作る

TaskUseCase

```
func getStorageReference() -> StorageReference? {  
    guard let uid = Auth.auth().currentUser?.uid else {  
        return nil  
    }  
    return storage.reference().child("users").child(uid)  
}
```

import FirebaseStorage

```
func saveImage(image: UIImage?, callback: @escaping ((String?) -> Void)) {  
    guard let image = image,  
          let imageData = image.jpegData(compressionQuality: 0.5),  
          let imageRef = getStorageReference() else {  
        callback(nil)  
        return  
    }  
}
```

```
let imageName = NSUUID().uuidString  
let metaData = StorageMetadata()  
metaData.contentType = "image/jpeg"
```

```
let ref = imageRef.child(imageName)  
ref.putData(imageData, metadata: metaData) { (metaData, error) in  
    guard let _ = metaData else {  
        print("画像の保存に失敗しました。。。😭")  
        callback(nil)  
        return  
    }  
    print("画像の保存が成功した！！！！！！")  
    callback(imageName)  
}
```


③写真をCloudStorageにアップする処理を作る

TaskCollection

3-2. TaskCollectionからTaskUseCaseの処理を呼ぶ

```
import FirebaseStorage
```

※UIImageを扱うのでUIKitに変更したいところだけど、
あとでStorage使うのでStorageにしておく

```
func saveImage(image: UIImage?, callback: @escaping ((String?) -> Void)) {  
    taskUseCase.saveImage(image: image) { (imageName) in  
        guard let imageName = imageName else {  
            callback(nil)  
            return  
        }  
        callback(imageName)  
    }  
}
```

③写真をCloudStorageにアップする処理を作る

AddViewController

3-4. 画像変更管理フラグを作って画像変更時にセットする

```
var isSetImage = false
```

```
// 撮影もしくは画像を選択したら呼ばれる
func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any]) {
    print("撮影もしくは画像を選択したよ!")

    if let pickedImage = info[.originalImage] as? UIImage {
        imageView.contentMode = .scaleAspectFit
        imageView.image = pickedImage.resize(toWidth: 300)
        isSetImage = true
    }
    // 表示した画面を閉じる処理
    picker.dismiss(animated: true, completion: nil)
}
```

③写真をCloudStorageにアップする処理を作る

AddViewController

3-5. タスク登録時の処理に画像UPを足しつつ、最低限リファクタリングする

```
// MARK: Action Method
@objc func tapSaveButton() {
    print("Saveボタンを押したよ!")
    guard let title = titleTextField.text else {return}
    if title.isEmpty {
        print(title, "😡titleが空っぽだぞ〜")
        HUD.flash(.labeledError(title: nil, subtitle: "😡 タイトルが入力されていません!!!"), delay: 1)
        return // return を実行すると、このメソッドの処理がここで終了する。
    }
}
```

```
var tmpTask = TaskCollection.shared.createTask()
if let index = selectIndex {
    tmpTask = TaskCollection.shared.getTask(at: index)
}
tmpTask.title = title
tmpTask.memo = memoTextView.text
if isSetImage {
    TaskCollection.shared.saveImage(image: imageView.image) { (imageName) in
        guard let imageName = imageName else {
            HUD.flash(.labeledError(title: nil, subtitle: "😡 保存に失敗しました"), delay: 1)
            return
        }
        tmpTask.imageName = imageName
        self.saveTask(tmpTask)
        print("🌟保存に成功したよ")
    }
} else {
    saveTask(tmpTask)
}
```

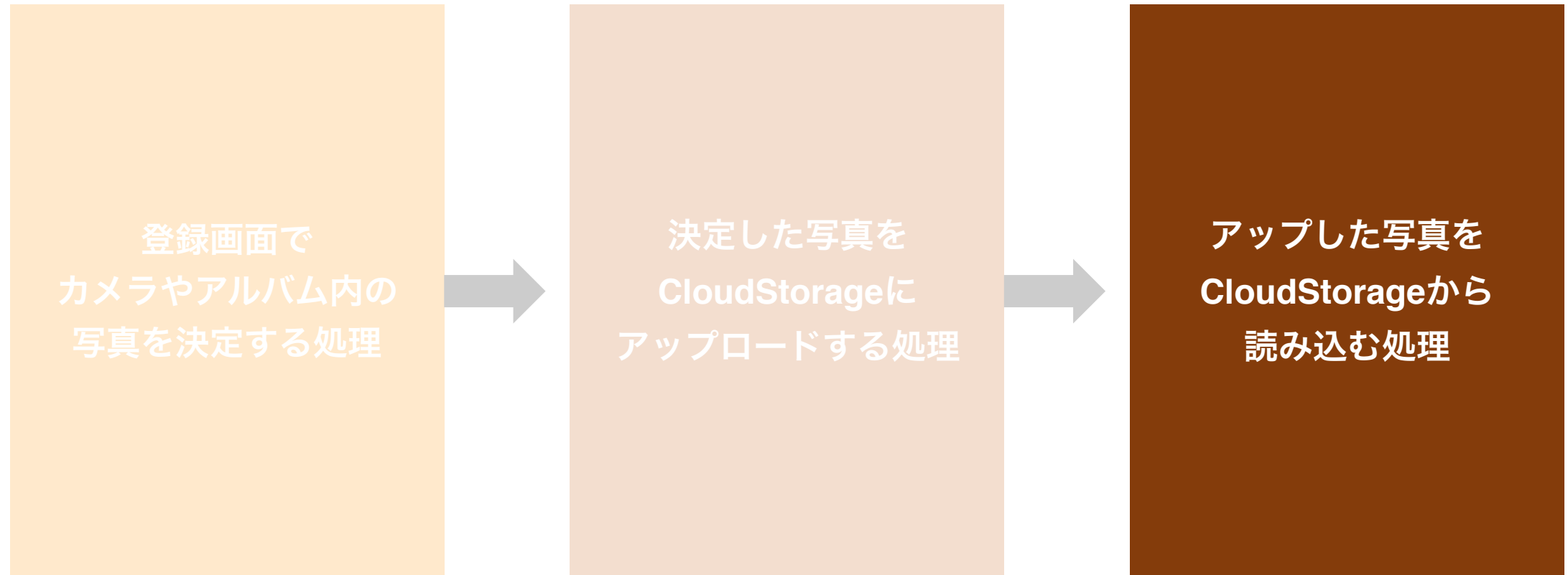
```
func saveTask(_ task: Task) {
    // ここで Edit か Add かを判定している
    if let index = selectIndex {
        task.updatedAt = Timestamp()
        TaskCollection.shared.editTask(task: task, index: index)
    } else {
        TaskCollection.shared.addTask(task)
    }
    HUD.flash(.success, delay: 0.3)
    // 前の画面に戻る
    navigationController?.popViewController(animated: true)
}
```

【一緒にやってみよう】

写真をダウンロードする処理を作ろう

※完成PJは配布するので
ついてこれなくなったら
手を止めて見ることに集中！

ここやります



やること（～読み込みまで）

- ①保存先のRefを取得する処理を作る
- ②FirebaseStorageUIを導入する
- ③AddVC読み込み時に表示する

①保存先のRefを取得する処理を作る

1-1. Refを取得する処理を作る

```
func getImageRef(imageName: String) -> StorageReference? {  
    return getStorageReference()?.child(imageName)  
}
```

TaskUseCase

1-2. Refを取得する処理を作る

```
func getImageRef(imageName: String) -> StorageReference? {  
    return taskUseCase.getImageRef(imageName: imageName)  
}
```

TaskCollection

②FirebaseStorageUIを導入する

PodFileに追加して…

```
pod 'FirebaseUI/Storage'
```

Podコマンド

```
$ pod install
```


③AddVC読み込み時に表示する AddViewController

3-1. ライブラリをimport

```
import FirebaseUI
```

3-2. selectedTaskの情報をセットしている処理を修正

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    setupMemoTextView()  
    setupNavigationBar()  
  
    // Editかどうかの判定  
    if let index = selectIndex {  
        title = "編集"  
  
        titleTextField.text = TaskCollection.shared.getTask(at: index).title  
        memoTextView.text = TaskCollection.shared.getTask(at: index).memo  
        if let imageName = TaskCollection.shared.getTask(at: index).imageName,  
            let ref = TaskCollection.shared.getImageRef(imageName: imageName) {  
            imageView.sd_setImage(with: ref)  
        }  
    }  
}
```

【(時間があれば)一緒にやってみよう】

画像を削除してみる

※完成PJは配布するので
ついてこれなくなったら
手を止めて見ることに集中！

やること

- ①画像を削除する処理を作る
- ②タスクの削除処理を色々修正

①画像を削除する処理を作る

TaskUseCase

1-1. 画像削除の処理を作る

```
func deleteImage(imageName: String?) {  
    guard let imageName = imageName, let ref = getImageRef(imageName: imageName) else { return }  
    ref.delete { (error) in  
        if let error = error {  
            print("画像の削除に失敗しました。。。😭", error)  
        } else {  
            print("画像の削除が成功した！！！！！！")  
        }  
    }  
}
```

②タスクの削除処理を色々修正 TaskUseCase

2-1.removeTaskの引数をtaskIdでなくTaskに変更して、deleteImageを呼ぶ

```
func removeTask(_ task: Task){  
    let documentRef = getCollectionRef().document(task.id)  
    documentRef.delete { (err) in  
        if let _err = err {  
            print("データ取得",_err)  
        } else {  
            self.deleteImage(imageName: task.imageName)  
            print("データ削除成功")  
        }  
    }  
}
```

②タスクの削除処理を色々修正 TaskUseCase

2-2. 引数が変わってエラーが出た箇所を修正

```
func removeTask(index: Int) {  
    taskUseCase.removeTask(tasks[index])  
    tasks.remove(at: index)  
    save()  
}
```

【(時間があれば)一緒にやってみよう】

読み込み時にくるくるを出してみる

※完成PJは配布するので
ついてこれなくなったら
手を止めて見ることに集中！

やること

- ①PKHUDを導入する
- ②入れたいところに入れる

①PKHUDを導入する

PodFileに追加して…

```
pod 'PKHUD'
```

Podコマンド

```
$ pod install
```

②入れたいところに入れていく

2-1. importする

```
import 'PKHUD'
```

2-2. 表示するとき

```
HUD.show(.progress)
```

2-3. 隠すとき

```
HUD.hide
```

多分入れたほうが良いところが
このアプリ内合計で3個くらいあると思います

各テーブルで話あって考えてみてください

【やってみよう】

このアプリの直すべきところを
洗い出してみよう

(切り口はUI/UX・足りない処理・…等、自由)

→ テーブルごとに発表

【次回までの課題】

自由

※注意点

GoogleService-Info.plistをプッシュしない

【最終回の予告】

リクエストがあったものを中心に
講義します。