iOS授業6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
2/1	Firebase Auth	・Firebaseファミリーの紹介 ・Firebaseでログイン機能を作成する
2/8	Cloud Firestore	・Firestoreでデータを保存・取得する
2/15	Firebase Storage	・Firebase Storageに画像を保存する
2/21	~~Cloud Functions~~ Tips TabBarController アプリの作成の手順 リファクタリング（GsTodo） エンジニアになる手っ取り早い方法	~~・サーバー側から処理を実施する~~ やって欲しいこと募集してます！！！ (何かあれば)

はじめに：今日やること

◆1.Firebaseについて

◆2.Firebaseセットアップ

◆3.FirebaseAuthを使ってみる

◆4.(時間があれば)WebKitViewを使ってみる

Firebaseについて

Firebaseとは？

・Googleが提供するmBaaS（mobile backend as a Service）

・2014年に買収、その後も買収&統合で機能強化

・モバイル開発に便利な機能がたくさん

・DBではリアルタイムで双方向に情報のやりとりが可能

スピードが求められるスタートアップにもってこい

一つのサービスを自分で全部構築するのはとても大変
学習コスト・メンテコストが高く、お金も時間も必要

クライアント(アプリ)

サーバーサイド

データベース

インフラ環境

スピードが求められるスタートアップにもってこい

一つのサービスを自分で全部構築するのはとても大変
学習コスト・メンテコストが高く、お金も時間も必要

クライアント(アプリ)

サーバーサイド

データベース

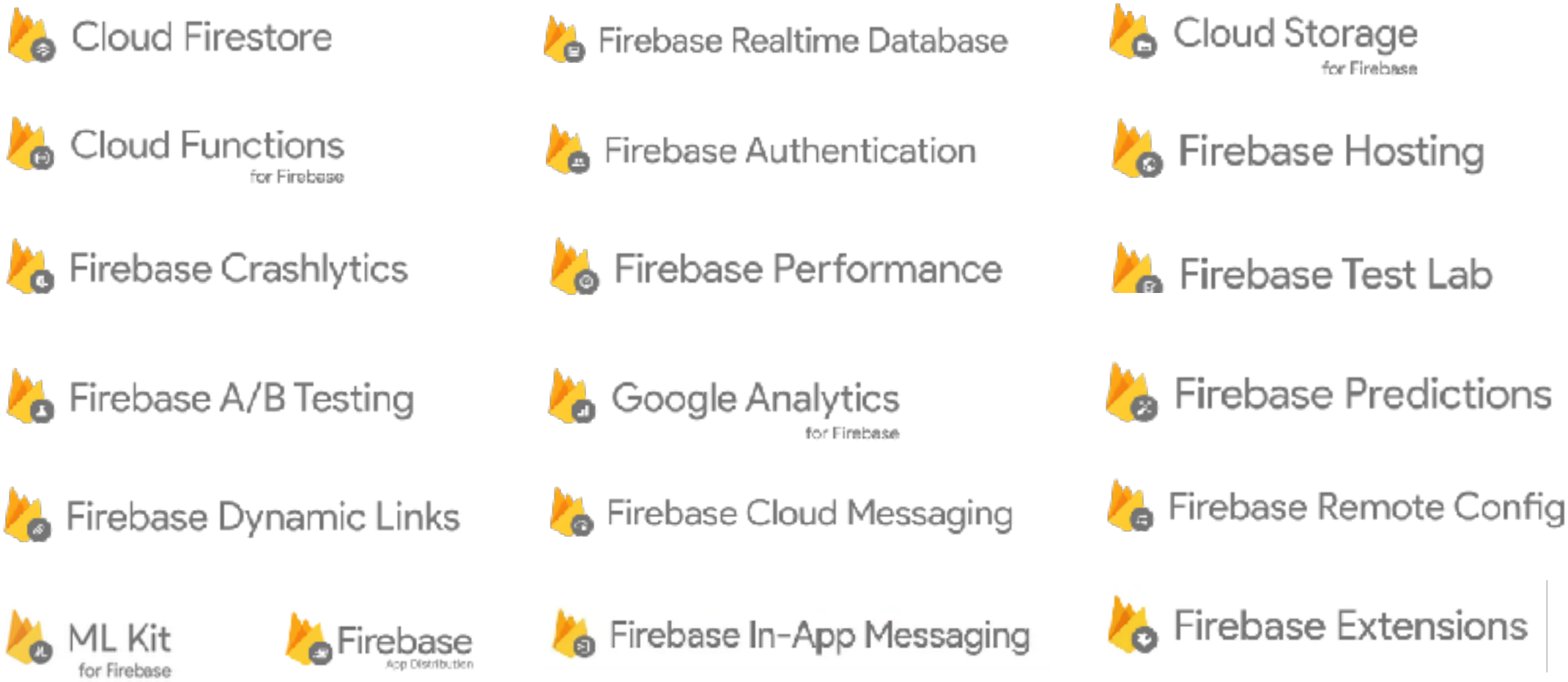
Firebase

インフラ環境

世界中の有名なアプリ・サービスで使われています

Firebaseの提供サービス

モバイル開発に便利な機能がたくさん!! ...ありすぎ!?



データベース系


Cloud Firestore　iOS授業でやる

今年β版から昇格した主力のデータベース。
複雑で階層的なデータの整理も得意で、青天井の自動スケール機能も備えている。
クエリはFRDよりましだが、複雑なものはサードパティーに頼る必要あり


Firebase Realtime Database　前やったやつ

元からあったデータベース。データを１つの大きなJSONとして扱う。
シンプルなデータはカンタンに保存することができてレイテンシも非常に低い。
複雑で階層的なデータを整理したり、クエリをかけたりすることが難しい。


Cloud Storage for Firebase　iOS授業でやる

写真や動画や音声など
ユーザーが作成したコンテンツを保管するストレージサービス。

ユーザー認証

Firebase Authentication　iOS授業でやる

バックエンド サービス、カンタンで使いやすい SDK、UI ライブラリ
などが用意されている

パスワード、電話番号、一般的なプロバイダ
（Google、Facebook、Twitter、GitHub、Microsoft、GameCenter）など
を使用した認証を行うことができる

サーバー処理

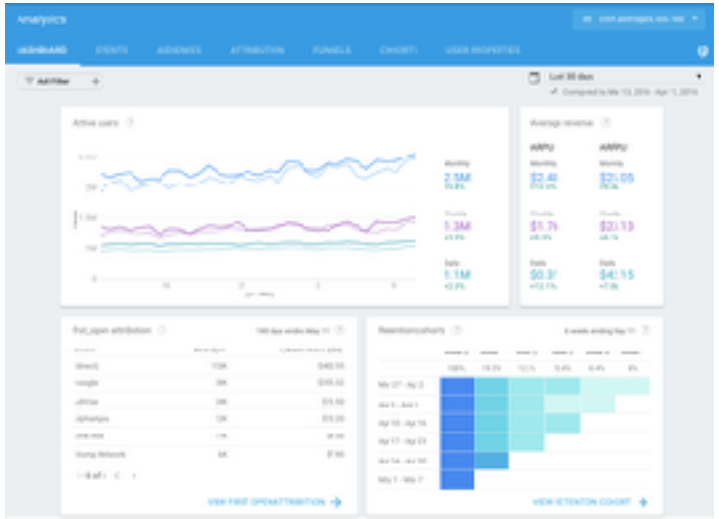
iOS授業でやらない

Googleのクラウド上に、サーバー側の処理をデプロイすることができる
ただ、コード自体はNode.jsとかで書く必要がある
対抗馬はAWSのLamdaなど

・データベースで●●に書き込まれたときにAという処理をする
・ユーザー作成時にBという処理をする
・Analyticsに●●というイベントが送信されたらCという処理をする
・毎日●時にDという処理を定期実行する

リサイズや翻訳など特定の処理であれば、コードなしで簡単に実現できる
Firebase Extentionというサービスがある(2019 Firebase Summit)
https://qiita.com/ryo2132/items/5e102136a1d7ad5d5e43

その他よく使いそうなもの



Google Analytics for Firebase

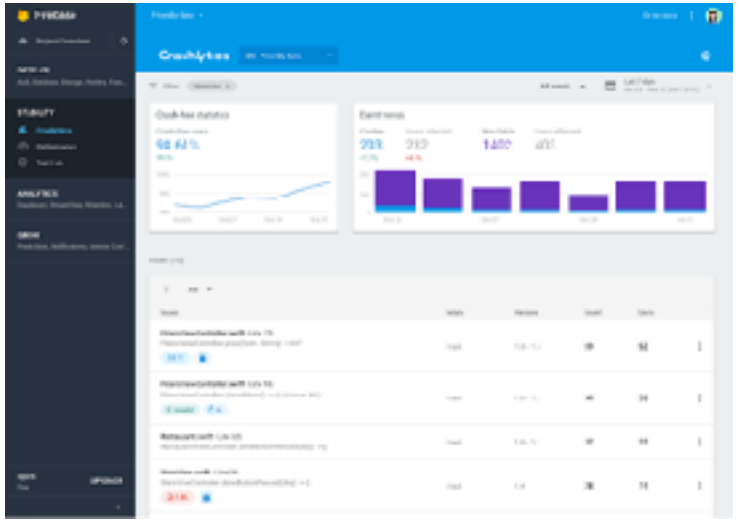
iOS授業でやらない

簡単なコードでイベントやユーザー属性を送信させて
分析・マーケティングに役立てることができる

元々FirebaseのAnalyticsとして独立していて、
GA側に送り出すには多少作業が必要だったが、
サービスがGAと統合されて分析がかなりしやすくなった

```
Analytics.logEvent(AnalyticsEventSelectContent, parameters: [
  AnalyticsParameterItemID: "id-\(title!)",
  AnalyticsParameterItemName: title!,
  AnalyticsParameterContentType: "cont"
  ])
```

その他よく使いそうなもの



Firebase Crashlytics　iOS授業でやらない

GoogleがTwitterのFabricを買収して追加された機能

クラッシュレポートを送信できる Slackなどへ送ると便利
そのエラーの回数やユーザー数など影響を見つつ対処が可能

dSYMをあげておけば、スタックトレースを見てエラーに対処できる
クラッシュと同時に何らかの情報を送信させることもできる
Analyticsを仕込んでおければ、直前のイベント・遷移なども参照できる

脱獄してる人もわかるｗｗｗ
※Fablicのβ版配信機能はApp Distribution側に継承

その他よく使いそうなもの

iOS授業でやらない

AppleのPush通知サーバーとのやりとりを仲介してくれるため、
証明書や素のTokenなどを扱わずに簡単にプッシュ通知が送れる
（なんで楽に送れるのか？：https://www.youtube.com/watch?v=A1SDBIViRtE）

Firebaseのコンソール(ダッシュボード)の画面から送ることはもちろん、
CloudFunctionsで処理を実行することもできる
例）誰かがFirestoreの●●という場所に書き込みをおこなった
→CloudFunctionsが反応→CloudFunctionsがPUSH通知のコードを実行
→誰かにメッセージがくる

その他よく使いそうなもの


Firebase Dynamic Links　iOS授業でやらない

ネイティブアプリの特定のコンテンツへ直接リンクできるディープリンクを生成してくれる。これによりWebページを開いたときに、アプリの該当するページを開いたり、メールやプッシュ通知にDynamic Linkを仕込んでおいて、実行したい処理を呼んだりできる。

更にすごいのが、iOS版のアプリ・Androidのアプリどちらでも同じことができるのと、未インストールの人にはダウンロードを促して、インストール完了後にアプリを起動して処理を実行させるといったこともできる。

その他よく使いそうなもの

Firebase Predictions　iOS授業でやらない

ユーザーが課金してくれるかどうか・アプリから離脱してしまうかどうかなどを機械学習で予測して、自動的にユーザーグループを作ってくれる。

Analyticsで作ったCVイベントに基づいた予測も可能。

例えば、離脱しそうな人にInAppMessagingで強めのオファーを出したり、エンゲージメントが高いユーザーへレビュー依頼してみたりとか。

その他よく使いそうなもの

 Firebase Remote Config　iOS授業でやらない

アプリのアップデートなしにアプリに読み込ませる値などを変えることができるので、UIのABテストや強制アップデート管理などに使うことができる

CloudFuntionsとCloudMessaging(サイレントプッシュ)を組み合わせることで、リアルタイムに伝播させることもできる

例）RemoteConfigで変更→
CloudFunctionsが反応→CloudFunctionsがサイレントプッシュを送る→
アプリ内の値を更新する→起動時にその値を見て処理を変える

その他よく使いそうなもの

Firebase In-App Messaging　iOS授業でやらない

メッセージ内容、配信対象の条件、スケジュール等を指定して、
アプリ内で簡単ポップアップメッセージを出すことができる

Analyticsのオーディエンス属性や
Predictionと組み合わせることで
複雑な施策の実行が可能

ここまでのおさらい

こんなことを学びました

①Firebaseについて

②Firebaseのサービスたち

Firebaseのプロジェクト作成
と
セットアップ

【一緒にやってみよう】

Firebaseのプロジェクトを作成して
Xcodeで使えるようにしよう

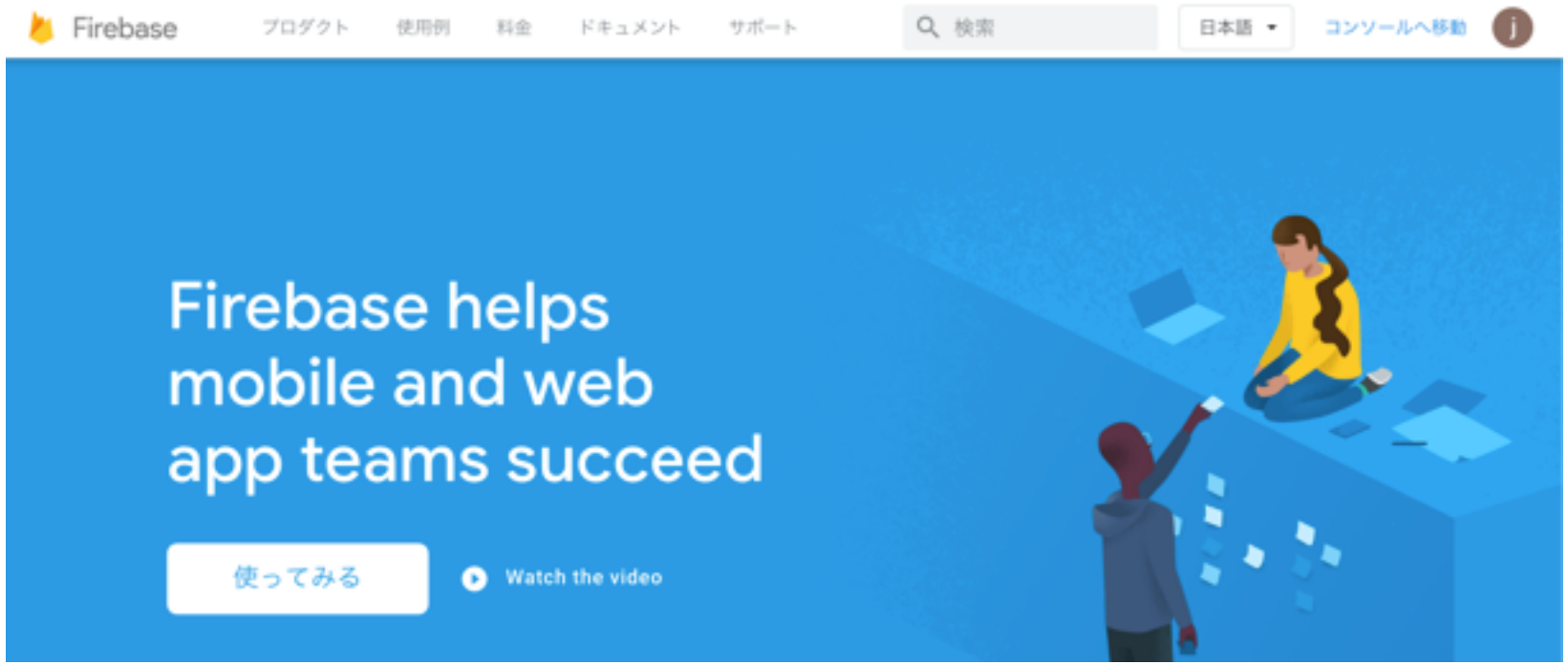
準備

配布した「GsTodo」に入っているPJを使います

BundleIDには各自のIDをセットしてください
“com.自分の名前.gstodo” がおすすめです

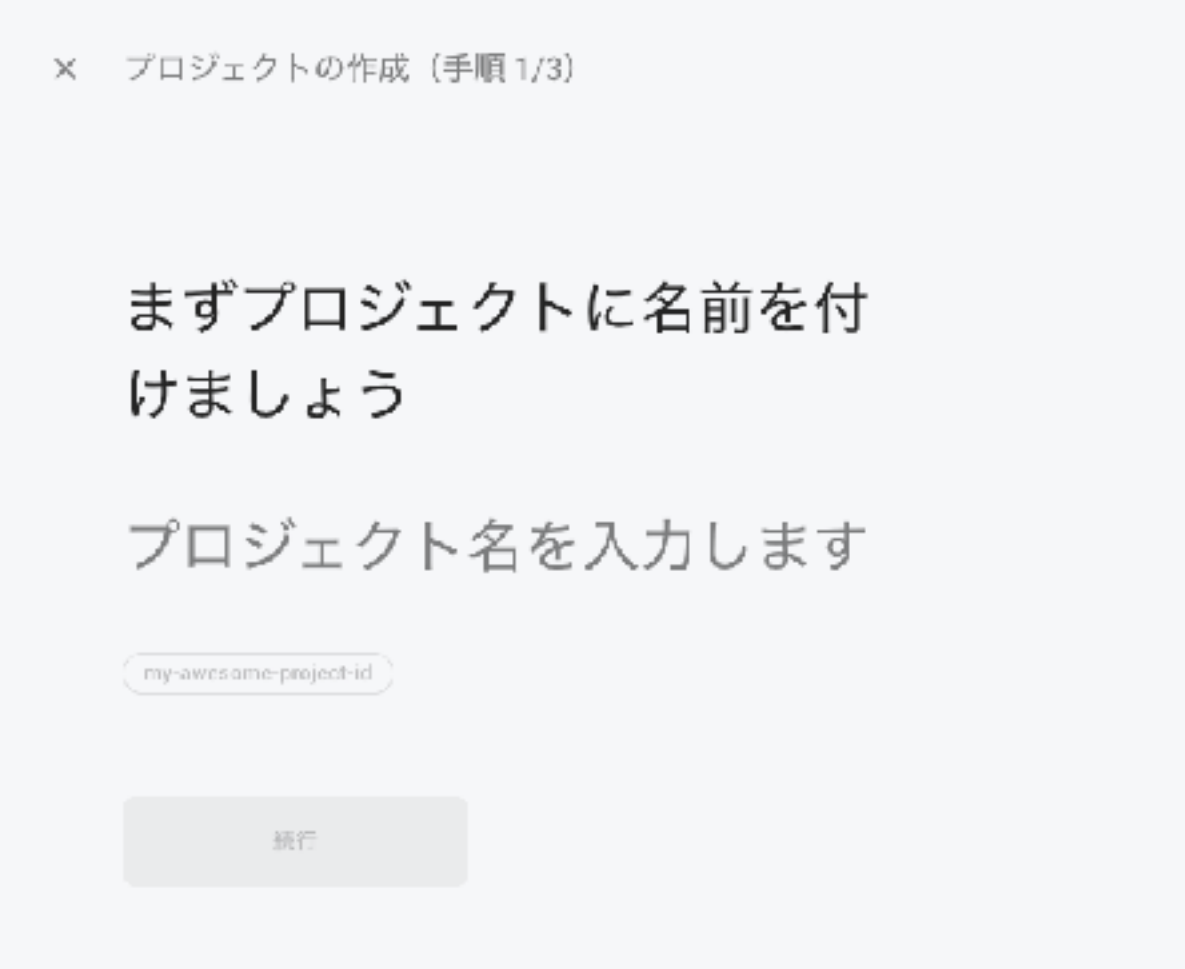
Firebaseサイトへアクセス

https://firebase.google.com/



プロジェクトを新規作成

プロジェクトの設定



・プロジェクト名：(自由に)

プロジェクトの設定

- アナリティクスは有効に

プロジェクトの設定

- Google へのデータ提供
  どっちでもいいですが、
  ONにしてます

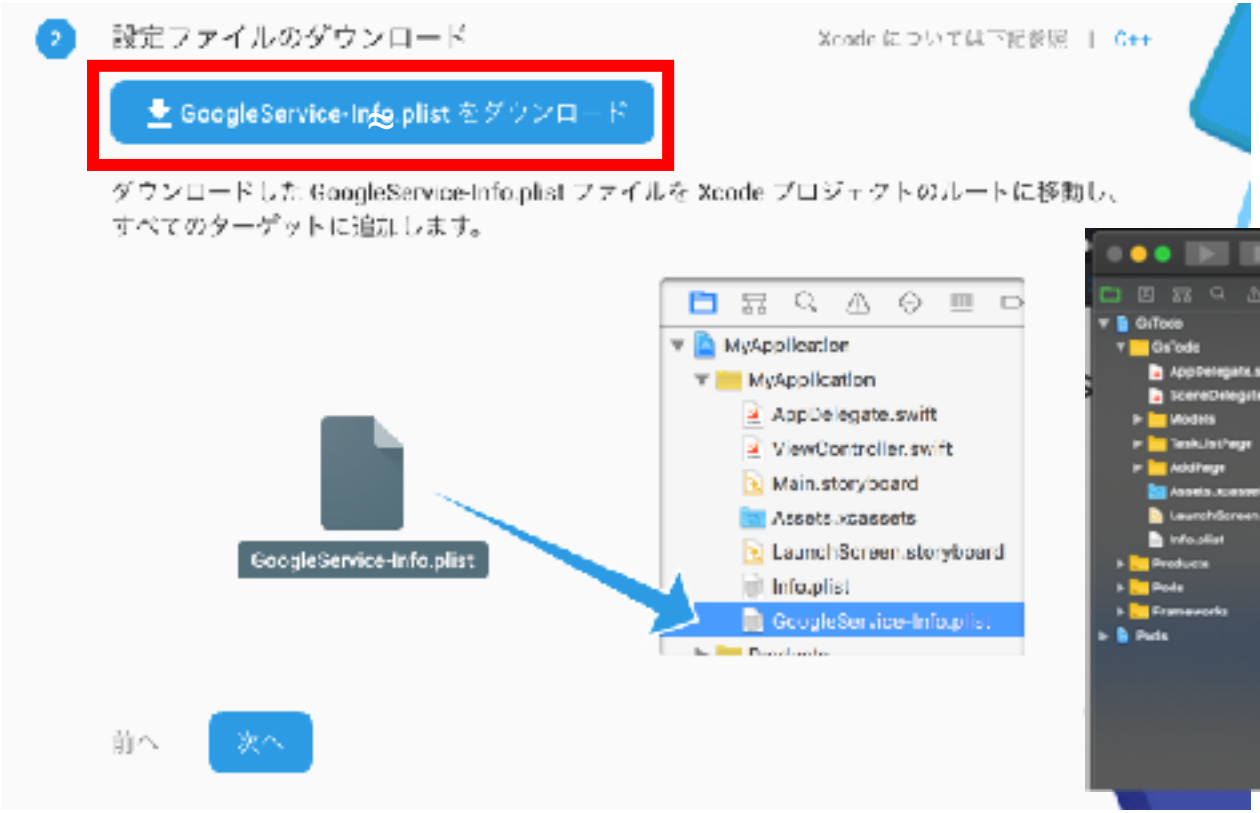
アプリを追加

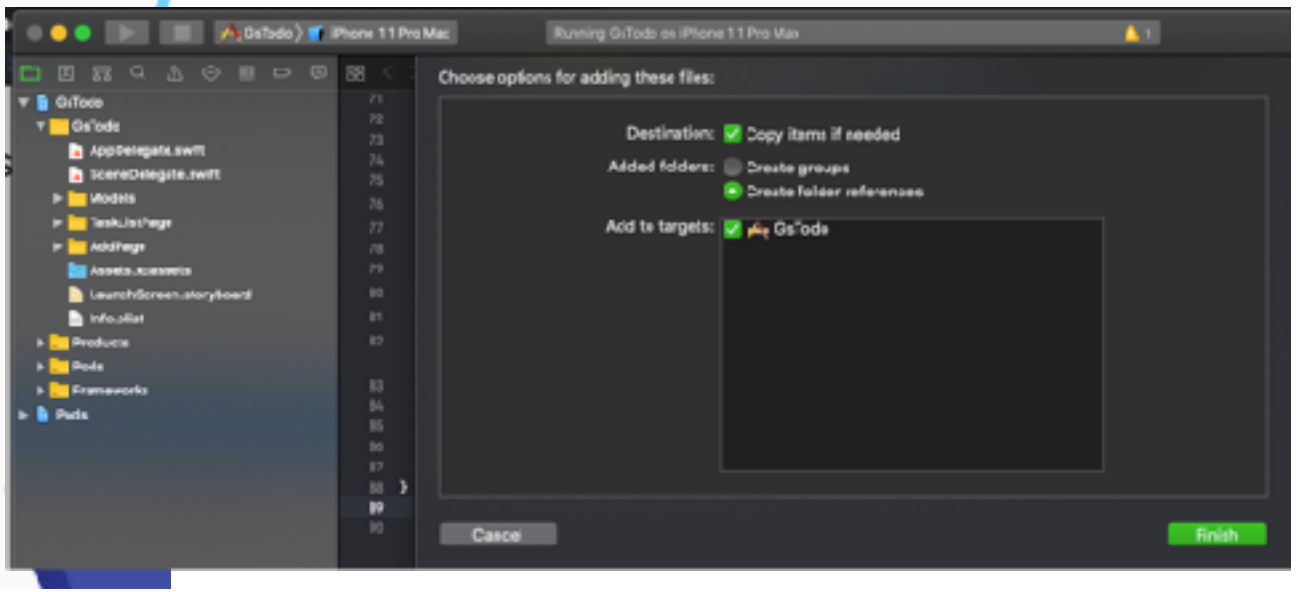

Firebase
Project Overview
開発
品質
アナリティクス
拡大
GsTodoTest
Firebase の機能、リサーチ、イベントに関する最新情報のメールを受け取る
GsTodoTest
Spark プラン
アプリに Firebase を追加して利用を開始しましょう
iOS
開始するにはアプリを追加してください

手順に沿ってアプリを設定①

- バンドルID：（Xcode使うBundleID）
- ニックネーム：（自由に）
- 「アプリを登録」

手順に沿ってアプリを設定②





設定ファイルをダウンロードして

ルート[プロジェクトフォルダの直下]に入れる

手順に沿ってアプリを設定③


```
target 'gstodo' do

  use_frameworks!

  # Pods for gstodo
  pod 'Firebase/Core'
  pod 'Firebase/Auth'

end
```

Podfileに「Firebase/Core」ついでに「Firebase/Auth」を追記して
「pod install」コマンドを打つ

手順に沿ってアプリを設定④

```
import UIKit
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication,
                     didFinishLaunchingWithOptions launchOptions:
                     [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        FirebaseApp.configure()
        return true
    }
```

AppDelegateでインポートをして、
didFinishLaunchingWithOptionsで「FirebaseApp.configure()」

補足：gitignoreへの追記

パブリックリポジトリで公開する場合は除外設定にしておく

Swift用のgitignoreの最後に
「GoogleService-Info.plist」と追記しておく

※チーム開発の場合は
　そもそもプライベートリポジトリにして共有する

ここまでのおさらい

こんなことを学びました

①FirebaseConsoleでプロジェクトを作成する方法

②FirebaseConsoleでアプリを追加する方法

③Firebaseのインストール作業

④Xcode側でのセットアップ作業

ログイン機能を作成する

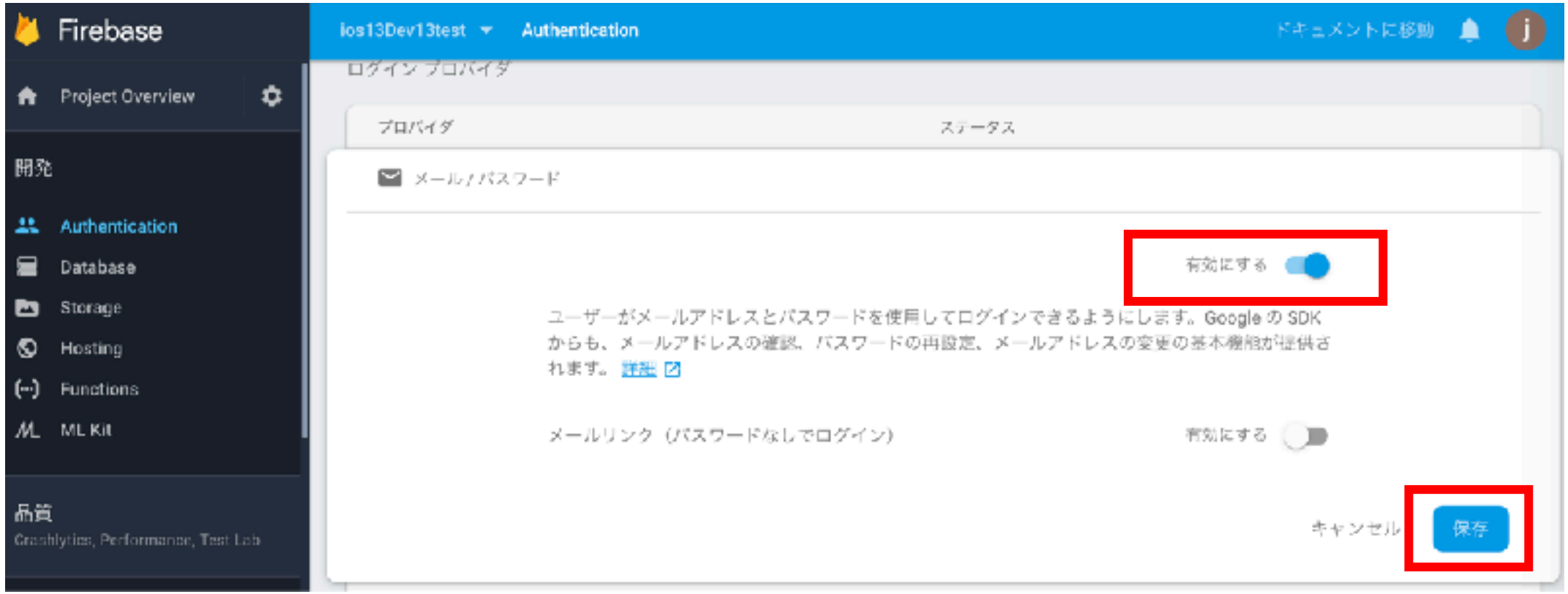
【一緒にやってみよう】

「Firebase Auth」を活用して
ログイン機能を作成しよう

※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！

事前準備

FirebaseConsoleでメール・パスワードを有効にする



やること一覧

①ログイン用のViewControllerを作成して画面を作る

②アラートを沢山使うのでExtensionで使いやすくする

③TextFieldの値が空だったらエラーを出す処理を作成

④サインアップ/サインイン処理を作成する

⑤エラー時の処理を作成する

⑥成功したときにTaskListへ遷移させる処理を作成する

⑦ログアウト処理を作成する

⑧ログイン状態を見てVCをを決める処理を作成

①ログイン用のViewControllerを作成して画面を作る

画面

LoginViewController

```swift
import UIKit

class LoginViewController: UIViewController {

    @IBOutlet weak var emailTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    @IBAction func tapSignUpButton(_ sender: Any) {
    }

    @IBAction func tapLoginButton(_ sender: Any) {
    }

    @IBAction func tapTermsButton(_ sender: Any) {
    }

}
```

①ログイン用のViewControllerを作成して画面を作る

最初に表示する画面を変える

SceneDelegate

```
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {
        guard let scene = (scene as? UIWindowScene) else { return }
        #warning("ここに次の記事のコードを記述する。http://bit.ly/2uwW1nz")
        window = UIWindow(windowScene: scene)
        window?.rootViewController = LoginViewController()
        window?.makeKeyAndVisible()
    }

    func sceneDidDisconnect(_ scene: UIScene) {
        // Called as the scene is being released by the system.
        // This occurs shortly after the scene enters the background, or when its session is discarded.
        // Release any resources associated with this scene that can be re-created the next time the scene connects.
        // The scene may re-connect later, as its session was not neccessarily discarded (see `application:didDiscardSceneSessions`
instead).
    }
```

②アラートを沢山使うのでExtensionで使いやすくする

UIViewController+Alert

```
import UIKit

extension UIViewController {
    func showErrorAlert(title: String) {
        let alertController = UIAlertController(title: “エラー",
                                  message: text , preferredStyle: .alert)
        let action = UIAlertAction(title: "OK", style: .default)
        alertController.addAction(OKAction)
        present(alertController, animated: true, completion: nil)
    }
}
```

これでUIViewControllerで使えるようになったので

試しにAddViewControllerのアラートを置き換えてみよう

③TextFieldの値が空だったらエラーを出す処理を作成

「tapSignUpButton」と「tapLogInButton」の中に
以下の処理を追記

LoginViewController

```
guard let email = emailTextField.text,
    let password = passwordTextField.text else { return }
if email.isEmpty {
    showErrorAlert(title:"メールアドレスを入力して下さい")
    return
}
if password.isEmpty {
    showErrorAlert(title: "パスワードを入力して下さい")
    return
}
```

④サインアップ/サインイン処理を作成する

LoginViewControllerでFirebaseAuthをインポートしたのち

Firebaseを使った登録・ログイン処理を作成する

LoginViewController

```
func emailSignUp (email: String, password: String){
    Auth.auth().createUser(withEmail: email, password: password) { (result, error) in
        if let _ = error {
            print ("😈登録失敗")
        } else {
            print ("! ⍰登録成功")
        }
    }
}
func emailSignIn (email: String, password: String){
    Auth.auth().signIn(withEmail: email, password: password) { (result, error) in
        if let _ = error {
            print ("😈ログイン失敗")
        } else {
            print ("! ⍰ログイン成功")
        }
    }
}
```

④サインアップ/サインイン処理を作成する

前ページで作った処理を
「tapSignUpButton」と「tapLogInButton」で呼ぶ

tapSignUpButtonで呼ぶ

LoginViewController

```
emailSignUp(email: email, password: password)
```

tapLogInButtonで呼ぶ

```
emailLogin(email: email, password: password)
```

⑤エラー時の処理を作成する

Firebaseから返ってくるエラーデータを使って
アラートのメッセージを変える（サインアップ）

LoginViewController

```
func signUpErrorAlert(_ error: Error){
    if let errCode = AuthErrorCode(rawValue: error._code) {
        var message = ""
        switch errCode {
        case .invalidEmail:       message =  "有効なメールアドレスを入力してください"
        case .emailAlreadyInUse: message = "既に登録されているメールアドレスです"
        case .weakPassword:       message = "パスワードは６文字以上で入力してください"
        default:                  message = "エラー: \(error.localizedDescription)"
        }
        showErrorAlert(text: message)
    }
}
```

エラーコード　https://firebase.google.com/docs/auth/ios/errors?hl=ja#firauth

⑤エラー時の処理を作成する

Firebaseから返ってくるエラーデータを使って
アラートのメッセージを変える（ログイン）

LoginViewController

```
func signInErrorAlert(_ error: Error){
    if let errCode = AuthErrorCode(rawValue: error._code) {
        var message = ""
        switch errCode {
        case .userNotFound:   message = "アカウントが見つかりませんでした"
        case .wrongPassword: message = "パスワードを確認してください"
        case .userDisabled:   message = "アカウントが無効になっています"
        case .invalidEmail:   message = "Eメールが無効な形式です"
        default:              message = "エラー: \(error.localizedDescription)"
        }
        showErrorAlert(text: message)
    }
}
```

エラーコード https://firebase.google.com/docs/auth/ios/errors?hl=ja#firauth

⑤エラー時の処理を作成する

「signUpErrorAlert」「LogInErrorAlert」をエラー時に呼ぶ

LoginViewController

```
func emailSignUp (email: String, password: String){
    Auth.auth().createUser(withEmail: email, password: password) { (result, error) in
        if let  error = error {
            self.signUpErrorAlert(_error)
        } else {
            print ("! ?登録成功")

        }
    }
}
```

```
func emailLogIn (email: String, password: String){
    Auth.auth().signIn(withEmail: email, password: password) { (result, error) in
        if let  error = error {
            self.LogInErrorAlert(_error)
        } else {
            print ("! ?ログイン成功")

        }
    }
}
```

⑥成功したときにTaskListへ遷移させる処理を作成する

移動する処理を作成する

LoginViewController

```
func presentTaskList () {
    let vc = UIStoryboard(name: "Main", bundle: nil)
    guard let vc = storyboard.instantiateInitialViewController() else {
        print(“viewControllerがないよ。。。”)
        return
    }
    vc.modalPresentationStyle = .fullScreen
    present(vc, animated: true)
}
```

⑥成功したときにTaskListへ遷移させる処理を作成する

「presentTaskList」 を成功時に呼ぶ

LoginViewController

```
func emailSignUp (email: String, password: String){
    Auth.auth().createUser(withEmail: email, password: password) { (result, error) in
        if let error = error {
            self.signUpErrorAlert(error)
        } else {
            self.presentTaskList()
        }
    }
}
```

```
func emailLogIn (email: String, password: String){
    Auth.auth().signIn(withEmail: email, password: password) { (result, error) in
        if let error = error {
            self.signInErrorAlert(error)
        } else {
            self.presentTaskList()
        }
    }
}
```

⑦ログアウト処理を作成する

TaskListTableViewControllerの
setupNavigationBar に以下のコードを追加

```
fileprivate func setupNavigationBar() {
    let rightButtonItem = UIBarButtonItem(barButtonSystemItem: .add, target: self, action:
        #selector(showAddScreen))
    navigationItem.rightBarButtonItem = rightButtonItem

    #warning("leftButton を作成して、 logout を実行する")
    let leftButtonItem = UIBarButtonItem(title: "logout", style: .plain, target: self, action:
        #selector(logout))
    navigationItem.leftBarButtonItem = leftButtonItem
}
```

⚠ LeftButton を作成して、 logout を実行する

TaskListTableViewController

⑦ログアウト処理を作成する

TaskListTableViewControllerでもFirebaseAuthをインポート後
ログアウトタップイベント処理を追記

TaskListTableViewController

```
@objc func logout(_ sender: Any) {
    do {
        try Auth.auth().signOut()
        let vc = LoginViewController()
        let sceneDelegate = view.window?.windowScene?.delegate as! SceneDelegate
        sceneDelegate.window?.rootViewController = vc
    } catch {
        print ("error:",error.localizedDescription)
    }
}
```

⑧ログイン状態を見てVCを決める処理を作成

AppDelegateのdidFinishLaunchingWithOptionsで
Firebaseのログインユーザーをチェックしてを変更する

SceneDelegate

```
if let _ = Auth.auth().currentUser {
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    guard let vc = storyboard.instantiateViewController) else {
        print(“viewControllerがないよ。。。“)
        return
    }
    window?.rootViewController = vc
} else {
    window?.rootViewController = LoginViewController()
}
```

ここまでのおさらい

こんなことを学びました

①サインアップの実装方法の例

②サインインの実装方法の例

③ログアウトの実装方法の例

④ログイン状態で画面を分ける実装方法の例

WebKit Viewを使用する

【一緒にやってみよう】

WebKit Viewを使って
サイトの情報を表示してみる

※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！

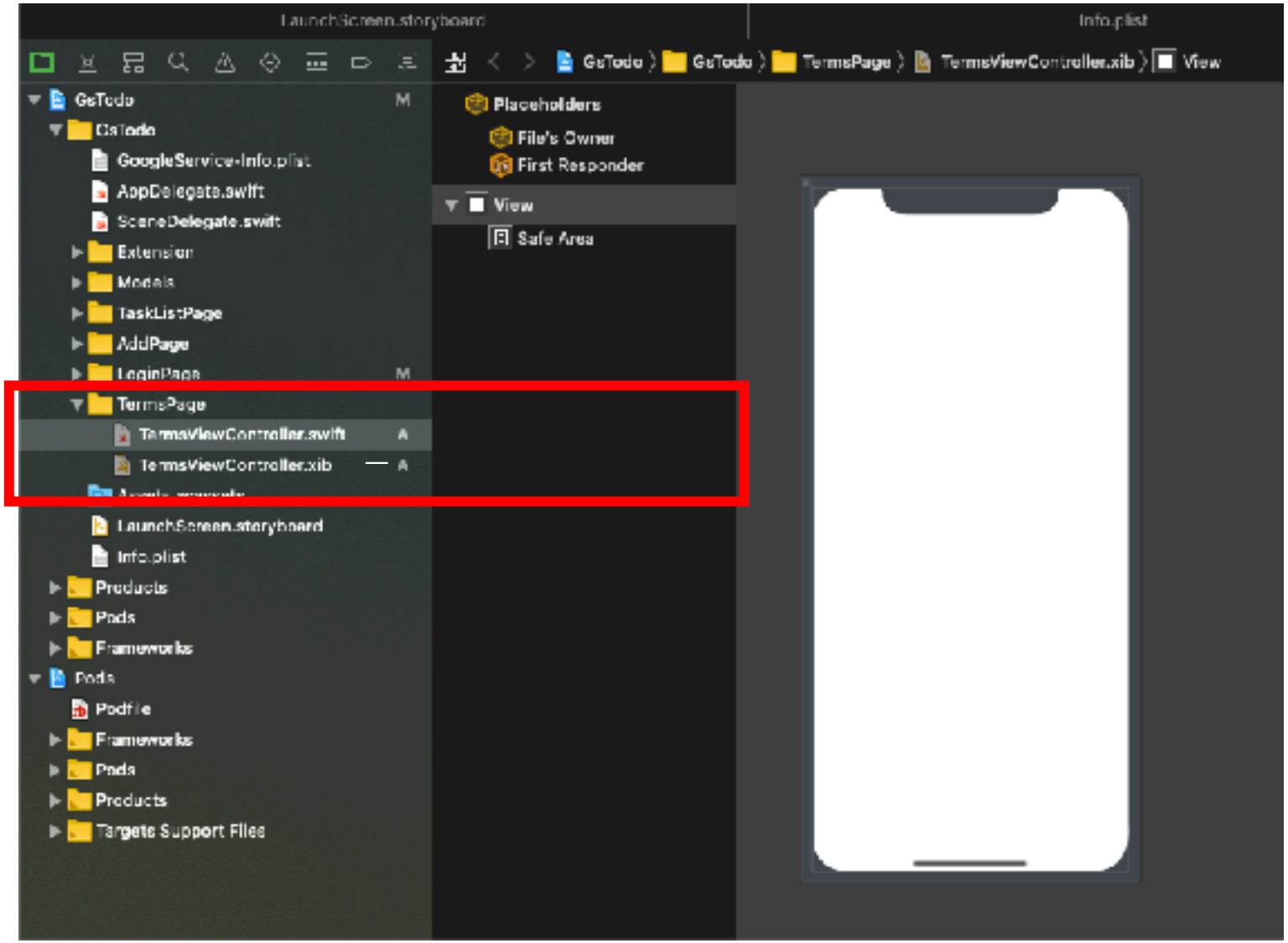
やること一覧

①利用規約用のVCを用意して画面作成

②WebViewで情報を読み込む

③サインインからの規約への遷移を作成する

①利用規約用のVCを用意して画面作成

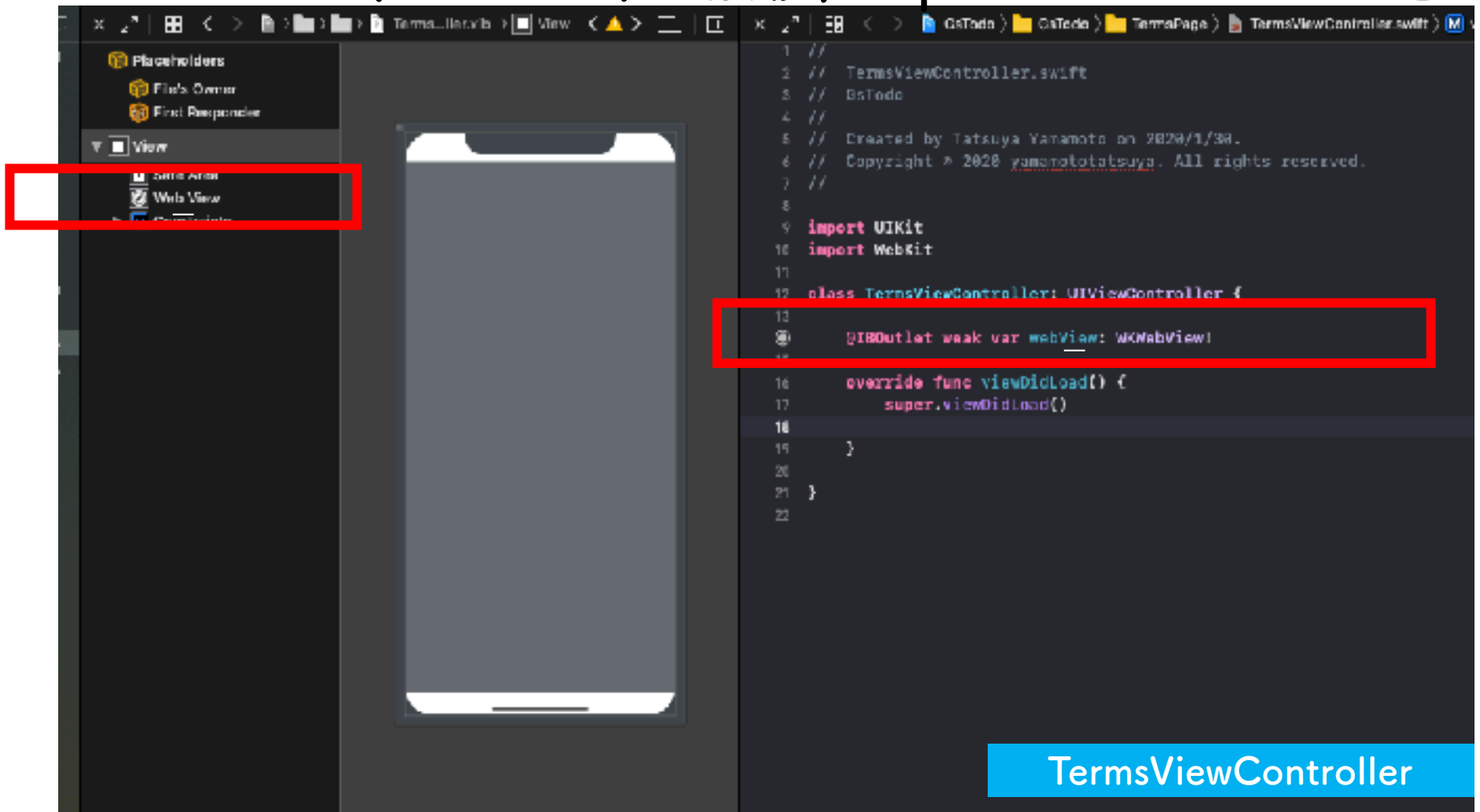


TermsViewControllerを作成する

①利用規約用のVCを用意して画面作成

WebKitViewを配置

WebKitView(IBOutlet)を接続, import WebKit を記述



②WebViewで情報を読み込む

ViewDidLoadでサイトを読み込み

```
override func viewDidLoad() {
    super.viewDidLoad()
    let termsURL = URL(string: "https://www.google.com/")
    let request = URLRequest(url: termsURL!)
    webView.load(request)
}
```

TermsViewController

③サインインからの規約への遷移を作成する

規約ボタンのイベントで遷移する処理を実装

LoginViewController

```
@IBAction func tapTermsButton(_ sender: Any) {
    let vc = TermsViewController()
    present(vc, animated: true)
}
```

補足：WebKitViewのDelegate

`WKNavigationDelegate`

遷移や読み込みの開始・完了・エラーなどを受け取りたい場合はこれを使う

```
func webView(_ webView: WKWebView, didCommit navigation: WKNavigation!) {
    print("WebビューがWebコンテンツの受信を開始したときに呼ばれる")
}
```

```
func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!) {
    print("ナビゲーションが完了したときに呼ばれる")
}
```

`WKUIDelegate`

新しいウィンドウを開いたり、

クリックして表示されるものの動作を補足したりする

ここまでのおさらい

こんなことを学びました

①WebKitViewの使い方

②WebKitViewのDelegate

【次回までの課題】

前回に引き続き自由

思いつかない人は他のプロバイダでの認証機能を作成してみる

※注意点
GoogleService-Info.plistをプッシュしない
.gitignore に書く

【次回の予告】

Firestoreを使ってデータを保存します

ここまでくれば、大体のアプリが作れる！？