Développement natif sous iOS

Cours n°6 - Gestion des données distantes

iOS

Gestion asynchrone

- Pour éviter que certaines opérations ne bloquent l'application, plusieurs threads sont lancés en parallèle.

- Celui gérant l'interface est le principal (*main thread*)

- Les opérations coûteuses (e.g. écriture de fichiers, accès réseau) sont lancés dans des threads de fond (*background threads*)

- La solution la plus répandue pour gérer plusieurs threads est l'utilisation d'une file d'exécution (*dispatch queue*)

Gestion asynchrone

- Seuls les threads lancés sur la file principale (*main dispatch queue*) peuvent modifier des éléments de l'interface graphique

- Les opérations lentes ne doivent jamais être lancées sur la file principale au risque de geler l'application

- Plusieurs files sont disponibles pour vos opérations avec différentes priorités (appelé *Quality of Service* ou QoS) :

    - `userInitiated` : Priorité la plus haute ; l'utilisateur attend les résultats de l'exécution de cette file

    - `utility` : Priorité moyenne ; tâches non essentielles

    - `background`: Priorité basse ; utilisée pour les sauvegardes ou opérations de nettoyage par exemple

Gestion asynchrone

```
lazy var fileImages = DispatchQueue(label: "images", qos: .utility)

// ...

fileImages.async {
    // Opération coûteuse

    DispatchQueue.main.async {
        // Mise à jour de l'interface une fois la
        // tâche réalisée.
    }
}
```

Gestion asynchrone

- Similaires aux files d'exécution, il existe des files d'opérations

- La différence est que les opérations peuvent être annulées et contrôlées (*monitoring*)

- La QoS est associée à la tâche et non à la file

- L'exécution d'une tâche peut dépendre du résultat d'autres tâches

- https://developer.apple.com/documentation/foundation/operationqueue

Gestion des données distantes

- En utilisant iCloud
    - Stockage de documents ou de données
- En passant par un webservice
    - Stockage des données que vous souhaitez
    - Développement et déploiement du webservice à votre charge ou utilisation d’un service existant

iCloud

- Service de stockage fourni par Apple
- Aussi appelé « *Ubiquitous store* » (Stockage ubiquitaire / omniprésent)
- Nécessite un compte développeur Apple (~100€ / an)
- L’espace iCloud est réparti entre celui de l’utilisateur (base de données privée) et celui de votre application (base de données publique)
- La base de données privée nécessite :
    - Que l’utilisateur soit connecté à son compte iCloud
    - Une autorisation de l’utilisateur

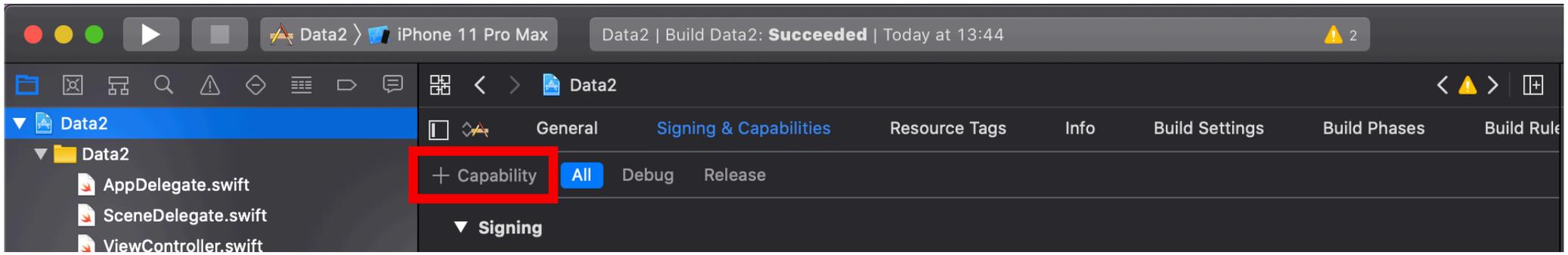
iCloud

- Côté SDK, utilisation de :
    - `NSUbiquituousKeyValueStore`
    - `CloudKit`
- Chaque application possède un « *Cloud Container* » (similaire à la *sandbox*)
- Il est possible de :
    - Tout stocker en ligne
    - Ou de garder des données en local et de synchroniser de temps à autres (notamment avec Core Data)

iCloud

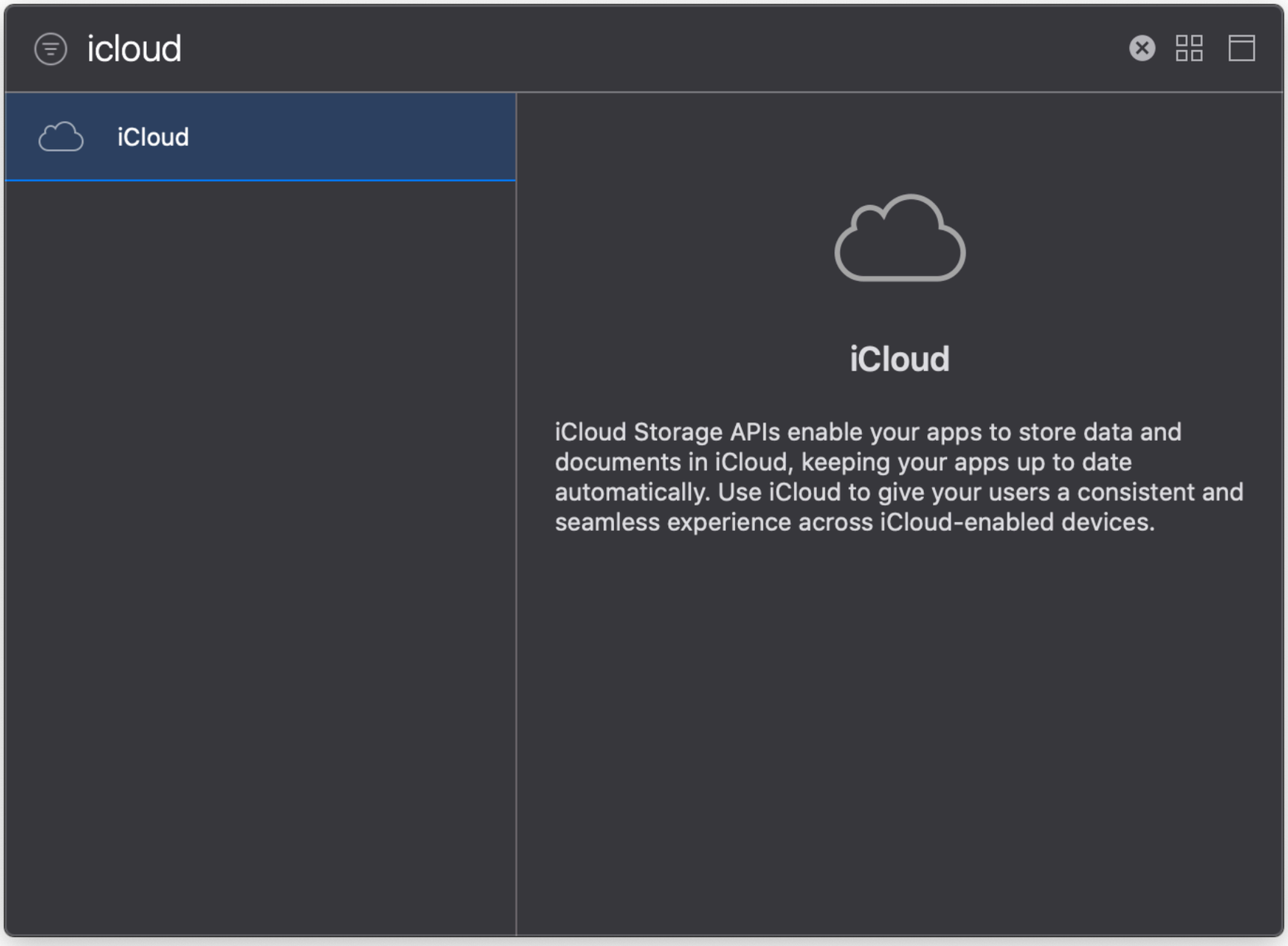
- Avantages :
    - Toute l’infrastructure est gérée par Apple
    - Synchronisation entre appareils et gestion des erreurs en cas de conflits
    - Envoi des données une fois le réseau disponible
- Inconvénients :
    - Difficilement utilisable pour d’autres plateformes (e.g. Android)
    - Limité en espace et en fonctionnalités

iCloud

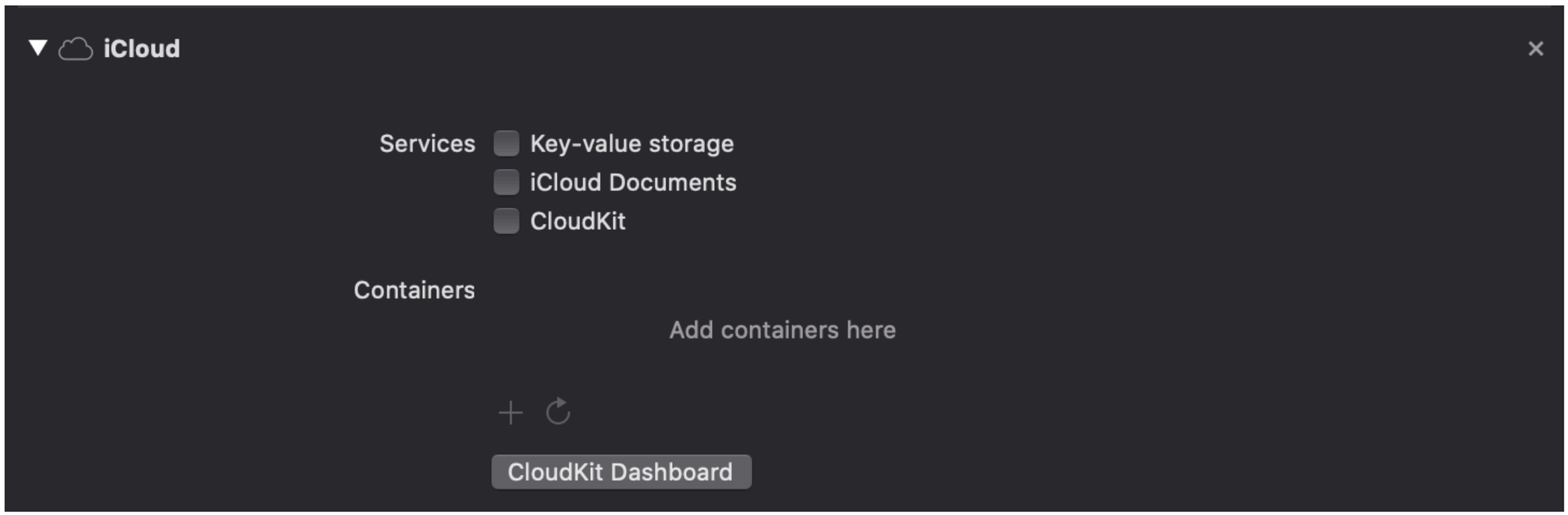


Il faut tout d’abord ajouter la capacité d’interagir avec iCloud dans notre application

iCloud


icloud
iCloud
iCloud
iCloud Storage APIs enable your apps to store data and documents in iCloud, keeping your apps up to date automatically. Use iCloud to give your users a consistent and seamless experience across iCloud-enabled devices.

iCloud



Les différents services qui peuvent être activés permettent de stocker des éléments de différentes natures

Les conteneurs, représentent les espaces dans lesquels sont stockés les éléments

Le bouton « CloudKit Dashboard » permet d’accéder au dashboard iCloud dédié à votre application

iCloud

(Stockage de valeurs simples)

Pour stocker une valeur :

```
NSUbiquitousKeyValueStore.default.set(0.9, forKey: "volume")
```

Pour récupérer une valeur :

```
NSUbiquitousKeyValueStore.default.double(forKey: "volume")
```

iCloud

(Stockage de structures)

- Pour gérer des structures autres que les types primitifs il faut :
    - Une instance de `CKDatabase`, représentant la base de données (publique ou privée)
        - Une instance de `CKRecord`, représentant la structure que l'on souhaite enregistrer
- Il est possible de stocker des images avec `CKAsset`
- Il est possible de réaliser des connections entre éléments en utilisant `CKReference`

iCloud

(Stockage de structures)

```
import CloudKit

let db = CKContainer.default().privateCloudDatabase
let record = CKRecord(recordType: "documents")

record["title"] = "Rapport de stage"
record["nbOfPages"] = 4

db.save(record) { (doc, error) in
    if error != nil {
        print("Une erreur est survenue")
    } else {
        print("Document sauvegardé dans iCloud")
    }
}
```

iCloud

- Il faut assurer une bonne gestion des erreurs de communication avec iCloud :
    - Mise à jour de l’interface
    - Réponse à l’erreur (nouvelle tentative, affichage d’une dialogue, etc.)
- Les erreurs peuvent survenir car :
    - L’utilisateur n’est pas connecté à son compte iCloud
    - Le réseau n’est pas disponible
    - Les données ne sont plus synchronisées (e.g. un autre appareil a mis à jour les données sur iCloud pendant l’envoi)
- https://developer.apple.com/documentation/foundation/icloud/icloud_error_codes
- https://developer.apple.com/documentation/cloudkit/ckerror/code

Utilisation d’un webservice

- Communication avec un service web existant ou développé par vos soins
- Peut être un service REST, GraphQL, peu importe
- Le format des réponses (JSON, texte, ou … XML) importe peu également

Utilisation d’un webservice

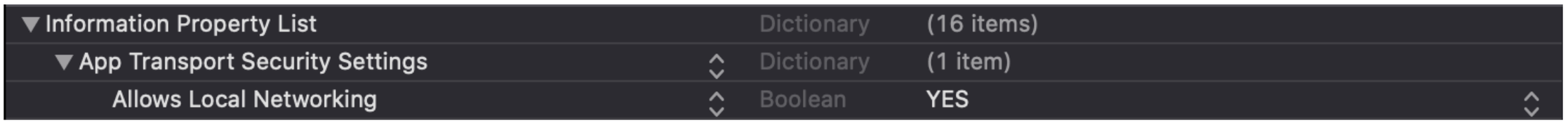
- Avantages :
    - Possibilité de synchroniser les données entre des appareils de différentes natures
        - Plus de possibilités qu’avec iCloud
- Inconvénients :
    - Vous devez maintenir le service (ce qui implique gérer un serveur ou s’assurer que le fournisseur est fiable) ou dépendre d’un service sur lequel vous n’avez pas la main

iOS et internet

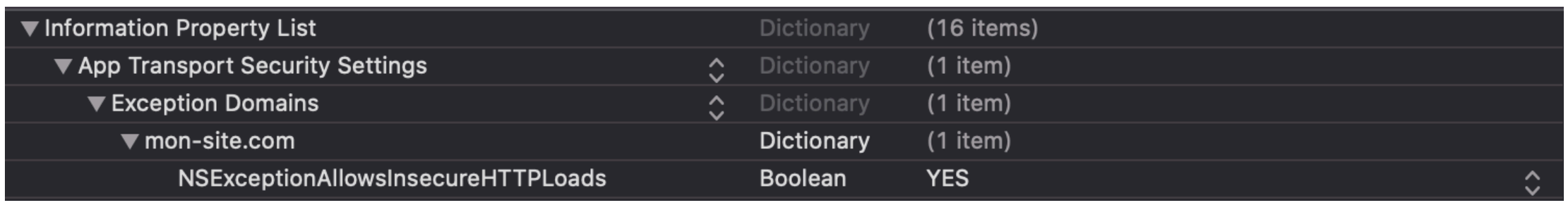
- Par défaut iOS ne supporte que les connexions sécurisées (via HTTPS) et par le biais du protocole TLS 1.2
- Il est possible, via le fichier `Info.plist`, de :
    - Spécifier une liste de domaines d’exception et spécifier que les connexions non sécurisées sont autorisées sur ces domaines
    - Autoriser les connexions locales (pratique pour le développement)

iOS et internet

Pour autoriser les connexions locales :



Pour autoriser un ou plusieurs sites via une connexion non sécurisée :



(Ne surtout pas préciser de port ou de `http://` devant le nom de domaine)

iOS et internet

- Sur émulateur, vous pouvez directement utiliser `localhost`, quelque soit le port

- Si vous utilisez un appareil branché à votre ordinateur :
  - Connectez le au même réseau que pour votre ordinateur
  - Utilisez l'adresse de votre machine sur le réseau local (en `192.168`)
  - Des limitations sur les ports accessibles peuvent être présentes selon votre configuration réseau

iOS et internet

- Le SDK iOS propose des facilités relativement bas niveau pour :
    - Construire des URLs (`URL`, `URLComponents`)
    - Créer des requêtes (`URLSession`, `URLRequest`)
    - Gérer le cycle de vie des tâches récupérant les données distantes (`URLSessionDataTask`)
- Transformer la réponse textuelle (e.g. en JSON) en objet Swift est à la charge du développeur

Réalisation de requêtes

- La gestion des requêtes dans une application iOS est relativement répétitive et longue
- Plusieurs projets rendent la tâche plus simple
    - Alamofire est relativement répandu et suffisamment puissant pour gérer la plupart des cas

Gestion des dépendances

- Installation de dépendances externes :
    - Pour faciliter le développement (pas besoin de réinventer la roue)
    - Pour partager du code entre différents projets
- Un gestionnaire de paquets simplifie l’utilisation et la mise à jour de ces dépendances

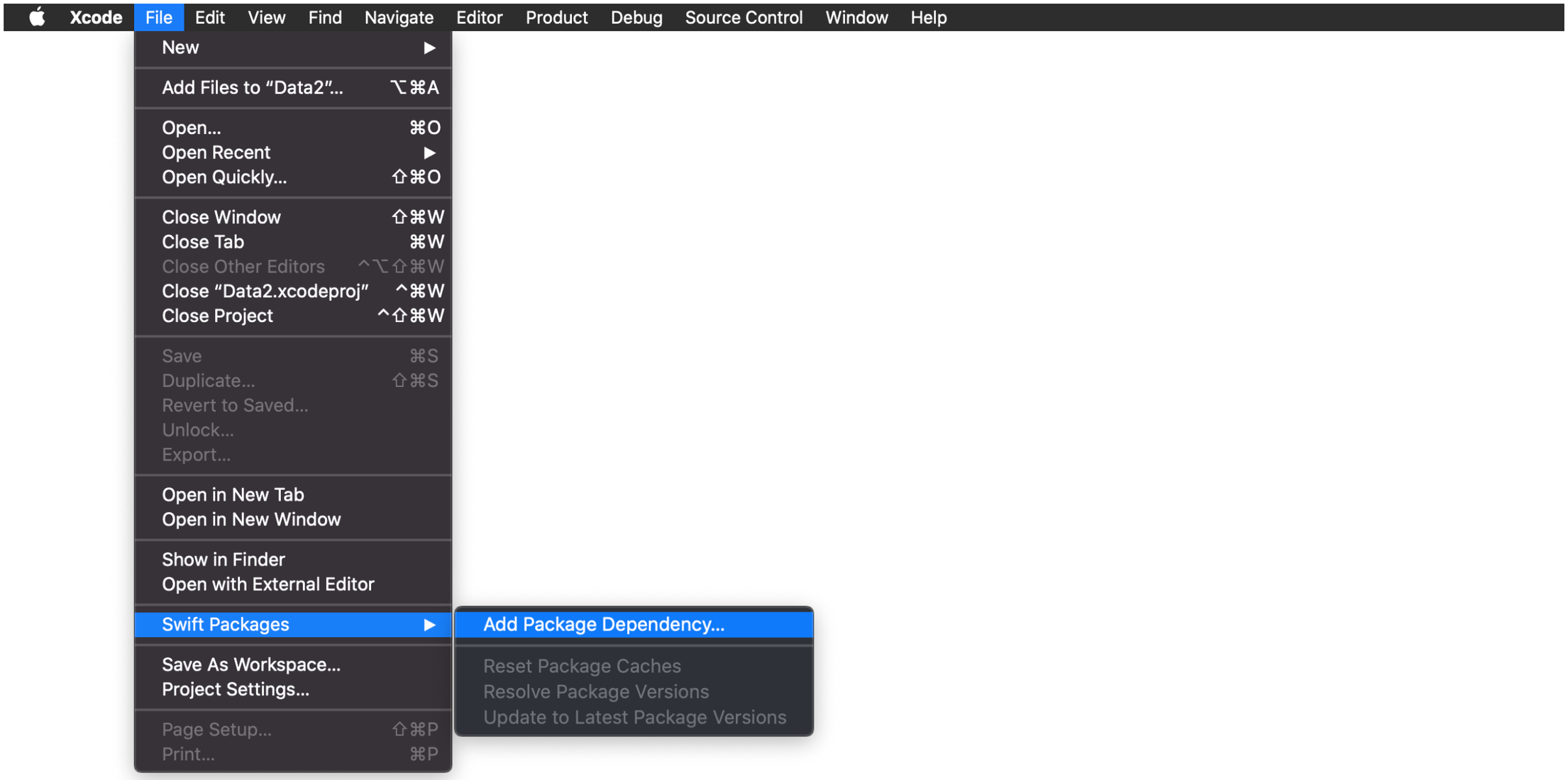
Gestion des dépendances

- Concept présent dans beaucoup d’autres langages :
    - Java (& Cie) : Maven, Gradle
    - PHP : Composer
    - C# (& .NET) : NuGet
    - etc.
- Plusieurs options existent :
    - Cartage
    - CocoaPods
    - Swift Package Manager (développé par Apple)

Swift Package Manager

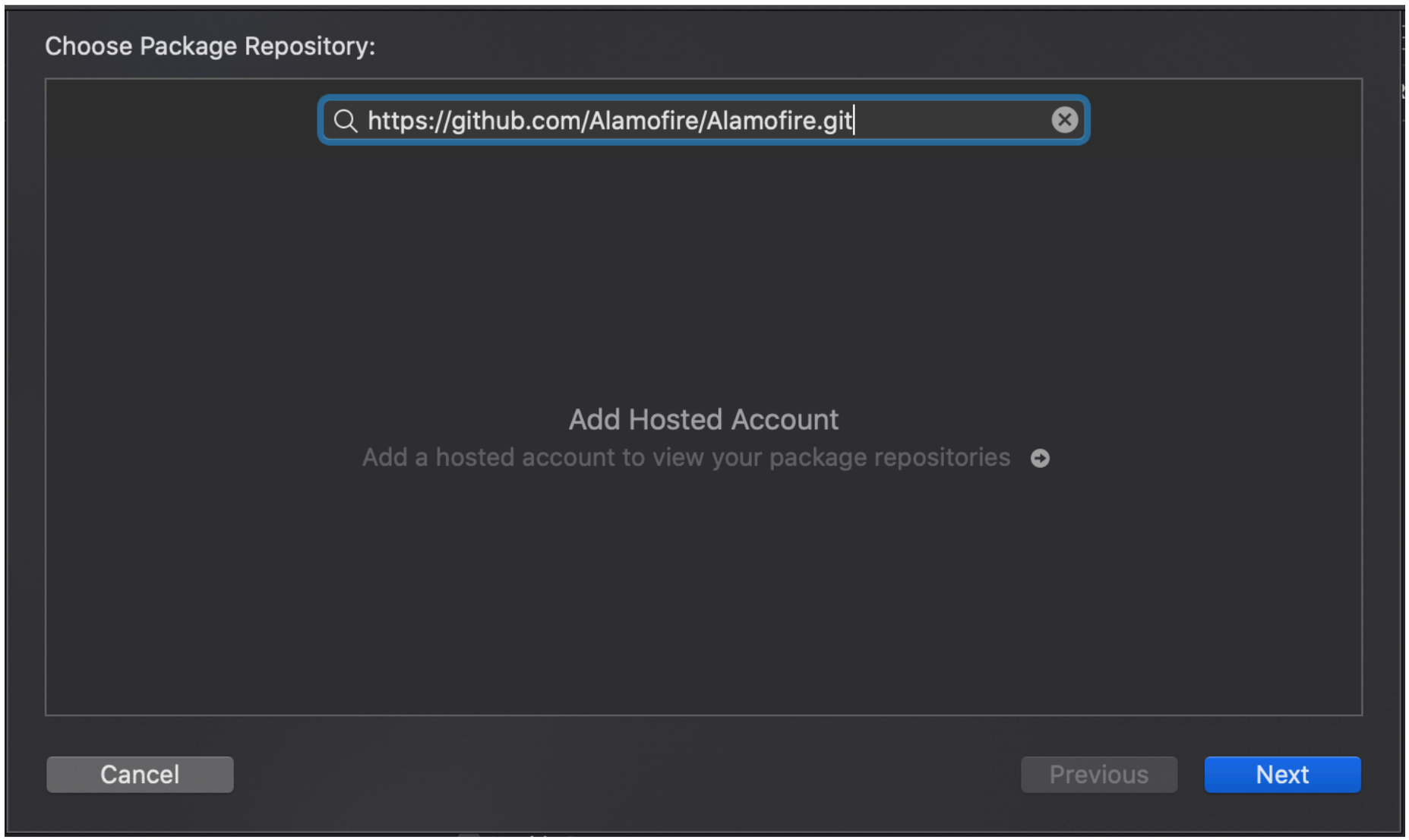
- Standard (recommandé par Apple)
- Fonctionne également sous Linux (contrairement à CocoadPods et Cartage)
- Les fichiers de dépendances sont écrits en Swift (contrairement à CocoadPods qui utilise Ruby)
- Supporté depuis Xcode 11

Swift Package Manager

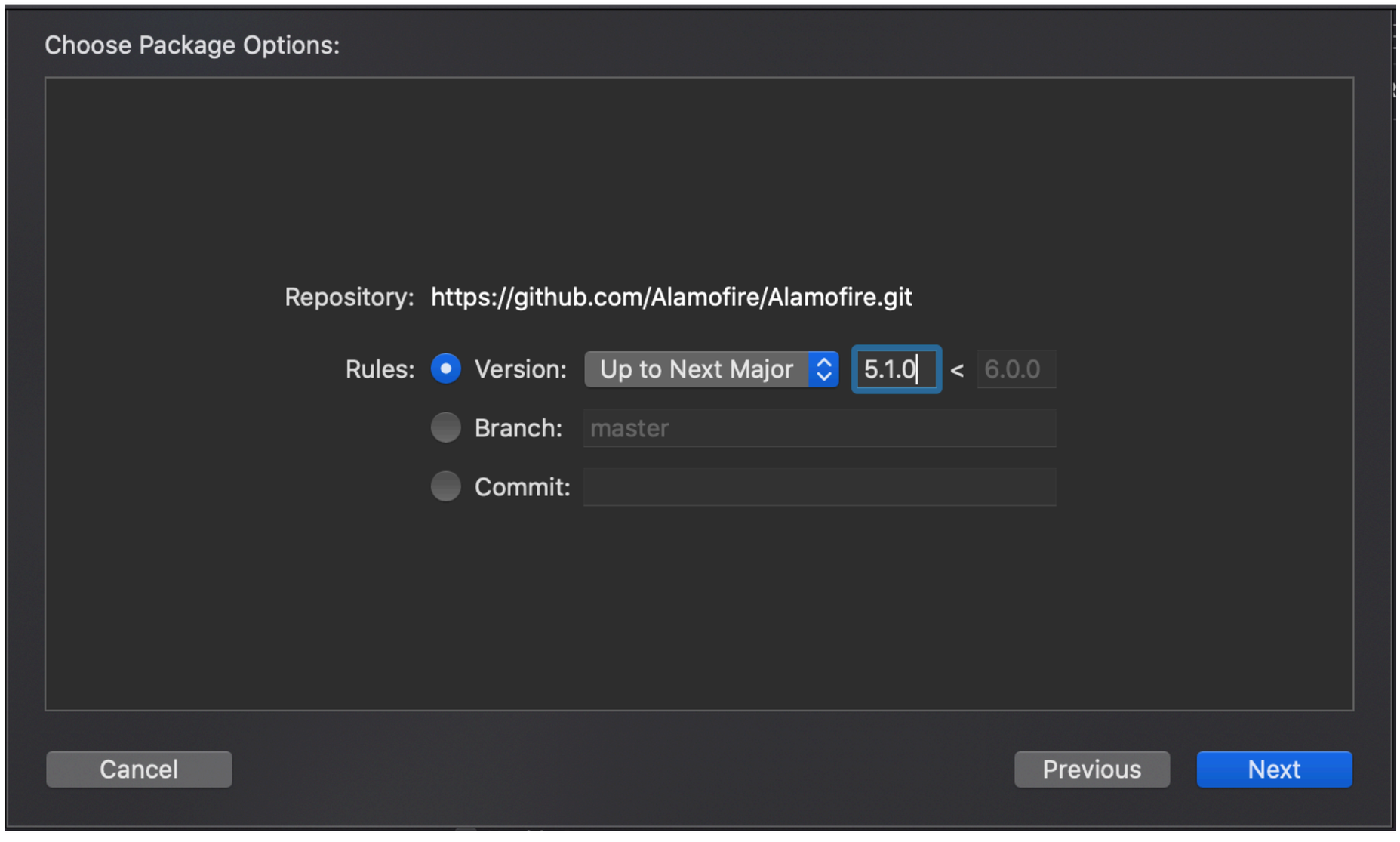


Il est possible d’ajouter directement vos dépendances externes depuis Xcode

Swift Package Manager



Swift Package Manager



Swift Package Manager

- Pour supprimer ou ajouter des dépendances :
    - Cliquer sur le nom du projet dans la panneau gauche
    - Puis cliquer sur le menu « Swift Packages »
- Il suffit de double cliquer sur une dépendance pour changer les contraintes de version depuis ce même menu

Alamofire

```swift
import Alamofire

// Requête GET
AF.request("https://mon-service.com/donnees").response { response in
    // ...
}
```

```swift
// Requête POST
let params = ["login": "jean", "password": "s3cr3t"]

AF.request("https://mon-service.com/login",
           method: .post,
           parameters: params,
           encoder: JSONParameterEncoder.default)

// Requête DELETE
AF.request("https://mon-service.com/article/1", method: .delete)
```

Alamofire

```
AF.request("...")
    .cURLDescription { description in
        print(description)
    }
```

Pratique pour tester votre requête dans un terminal avec la commande `curl`

Notifications

- Les notifications peuvent être de deux types :
    - Locales : l'application les crée
    - Distantes (ou *push notifications*) : un serveur distant (développé par vos soins par exemple) les envoie
- Le framework gérant les notifications du SDK iOS s'appelle `UserNotifications`.

Notifications

- Pour créer des notifications, il faut tout d'abord demander l'autorisation
- Plusieurs fonctionnalités liées aux notifications peuvent être demandées :
    - `alert` : afficher une notification sur l'écran
    - sound : émettre un son quand une notification est créée
    - badge : mettre à jour l'icône de l'application avec un badge indiquant un nombre de notifications (e.g. nombre de messages non lus)
- Les notifications ne sont pas affichées si l'application est ouverte
- https://developer.apple.com/documentation/usernotifications/asking_permission_to_use_notifications

Notifications

(Demande d’autorisation)

```
let center = UNUserNotificationCenter.current()

center.requestAuthorization(options: [.alert, .sound]) { granted, error in
    if let error = error {
        // Gestion de l'erreur
    }

    if granted {
        // Autorisé
    } else {
        // Non autorisé
    }
}
```

Notifications locales

- Le SDK propose plusieurs facilités pour créer des notifications de différentes natures :
    - Par rapport au calendrier (e.g. tous les mardis)
    - Selon un intervalle de temps (e.g. dans 1 heure)
    - Selon une position géographique (e.g. dès que l'on se trouve dans un autre pays)
- https://developer.apple.com/documentation/usernotifications/scheduling_a_notification_locally_from_your_app

Notifications locales

```
// Contenu de la notification
let content = UNMutableNotificationContent()
content.title = "Nouveau message"
content.body = "Eh bien le bonjour !"

// Dans 30 secondes
let trigger = UNTimeIntervalNotificationTrigger(
    timeInterval: 30,
    repeats: false
)

// Demande d'envoi de la notification
let request = UNNotificationRequest(
    identifier: UUID().uuidString,
    content: content,
    trigger: trigger
)

center.add(request) { error in
    if error != nil {
        // Gestion de l'erreur
    }
}
```

Notifications distantes

- Nécessitent un compte développeur (même pour le développement en local)

- Nécessitent la création d’un service communiquant avec APNs (Apple Push Notification service)
    - Votre service doit stocker un token, unique par application et par appareil

    - Votre application doit donc demander et envoyer ce token à votre service

- https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

Notifications distantes

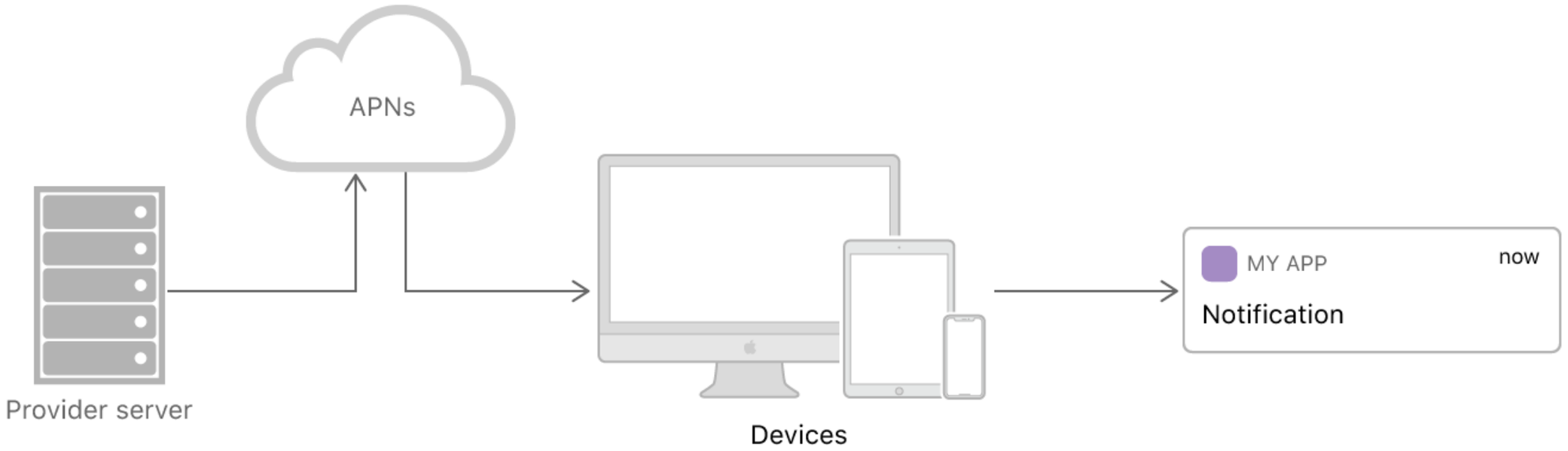
```swift
// Demande du token
func applicationDidFinishLaunching(_ application: UIApplication) {
    UIApplication.shared.registerForRemoteNotifications()
}

// Lors d'un succès
func application(_ application: UIApplication,
                 didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
    // Envoi du token à votre web service
}

// Lors d'un échec
func application(_ application: UIApplication,
                 didFailToRegisterForRemoteNotificationsWithError error: Error) {
    // On stocke un marqueur pour tenter ultérieurement
}
```

Code à placer au niveau de votre `AppDelegate`

Notifications distantes

- Conseils pour le stockage du token :
    - Prévoir la possibilité de stocker plusieurs tokens par utilisateur (un par appareil)
    - Ne jamais stocker le token en local ; Apple re-génère le token dans certaines situations
    - Quand une erreur a lieu lors de la demande du token, définir une variable pour re-tenter ultérieurement
- Conseils pour les notifications :
    - Ne jamais stocker de données sensibles dans la charge utile (i.e. la représentation d’une notification)

Notifications distantes



Notifications distantes

- Votre service doit supporter HTTP/2
- La connexion à votre service doit être sécurisée (utilisation de TLS 1.2 minimum)
- La communication avec APNs se fait au format JSON

Notifications

- Pour aller plus loin :
  - Création de notifications avec actions possibles
  - Création de notifications distantes
  - Gestion des requêtes vers APNs