

# APIs et fonctions avancées

Un petit aperçu de ce qui vous attends

# APIs et fonctions avancées

- ✦ Utiliser les capteurs
- ✦ Appels asynchrones
- ✦ Accéder au réseau
- ✦ Parsing JSON
- ✦ Animer son interface
- ✦ Outils tiers utiles

# Utiliser les capteurs

# Capteurs

- ✦ Selon les appareils, différents capteurs sont à notre disposition
  - ✦ GPS / Glonass / Magnétomètre
  - ✦ Gyroscope / Baromètre / Accéléromètre
  - ✦ Capteurs photo
  - ✦ Touch ID
  - ✦ etc.
- ✦ Pensez à tester la présence d'un capteur avant de l'utiliser !

# Géolocalisation

- ✦ Permet de déterminer la position et la direction de l'utilisateur
- ✦ L'utilisateur doit donner son accord !
- ✦ Utilise divers capteurs selon la précision demandée
  - ✦ Puce cellulaire, wifi, GPS

# Géolocalisation : autorisation

- ✦ Déclarer l'utilisation dans le fichier Info.plist
  - ✦ App au premier plan
  - ✦ En permanence
- ✦ Vérifier l'acceptation de l'utilisateur, ou faire la demande

# Géolocalisation : utilisation

- ✦ Framework Core Location requis
- ✦ Utilisation de la classe CLLocationManager et d'un délégué
  - ✦ Configurer le manager
  - ✦ Définir le délégué
  - ✦ Activer la géolocalisation

# Géolocalisation : être un bon citoyen

- ✦ Attention à la vie privée et à la loi
- ✦ Ne demander que les autorisations nécessaires
- ✦ Prévoir le cas où l'utilisateur refuse
- ✦ Demander la précision correspondant aux besoins
- ✦ Stopper la localisation dès que possible pour limiter l'impact énergétique



# Mouvement

- ✦ L'accéléromètre permet de connaître l'orientation dans l'espace de l'appareil et les mouvements (pas) de l'utilisateur
- ✦ Le gyroscope permet d'obtenir les mouvements dans l'espace de l'appareil
- ✦ Le baromètre permet de connaître le déplacement en hauteur (changement d'altitude)
- ✦ La classe CMMotionManager de Core Motion permet d'accéder aux informations de ces capteurs

# Photo

- ✦ UIImagePickerController permet de gérer simplement la prise de photos, ou la récupération de photos depuis la bibliothèque de l'utilisateur.
  - ✦ Créer un UIImagePickerController et le configurer
  - ✦ Lui définir son delegate
  - ✦ Récupérer les données via le delegate
- ✦ AV Foundation permet une récupération et un traitement plus précis du flux vidéo issu du capteur, mais contre une plus grande complexité de mise en oeuvre



# Photo : Exemple

- ✦ Créer un UIImagePickerController et le configurer

```
let imagePickerController = UIImagePickerController()  
imagePickerController.sourceType = .photoLibrary  
imagePickerController.delegate = self  
present(imagePickerController, animated: true, completion: nil)
```

- ✦ Récupérer les données via le delegate

```
func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [String : Any]) {  
    let image = info[UIImagePickerControllerOriginalImage]  
    //Use the image  
}
```

# Touch ID

- ✦ Utilisation du framework Local Authentication
- ✦ Le framework préserve la sécurité de l'utilisateur
  - ✦ L'application n'a jamais accès à l'empreinte digitale
  - ✦ On obtient simplement une réponse positive ou négative sur l'authentification de l'utilisateur



# Touch ID : Exemple

```
let context = LAContext()
var authError: NSError?
let myLocalizedString = "<#String explaining why app needs authentication#"

if context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error: &authError) {
    context.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, localizedReason:
myLocalizedString, reply: { (success, error) -> Void in

        guard success else {
            // User did not authenticate successfully, look at error, take appropriate action
            return
        }
        // User authenticated successfully, take appropriate action

    })
} else {
    // Could not evaluate policy; look at authError and present an appropriate message to user
}
```

# Appels asynchrones

# Pourquoi ?

- ✦ L'interface d'iOS s'exécute sur le thread principal
- ✦ Une longue tâche sur ce thread bloque l'interface
  - ✦ Si une application est bloquée plus de quelques secondes, elle est stoppée par iOS
- ✦ Les tâches longues doivent être effectuées en arrière plan
  - ✦ Un moyen simple d'y parvenir est d'utiliser Grand Central Dispatch
  - ✦ Plus simple que de gérer des threads et optimisé pour chaque device

# Dispatch

- ✦ GCD utilise des blocs qui sont dispatchers sur des files.
- ✦ Les files sont gérées par GCD et optimisées en fonction de la charge et du nombre de coeurs du CPU
- ✦ La fonction C `dispatch_async()` permet de dispatcher des blocs sur les files

```
dispatch_async(dispatch_queue_t queue, dispatch_block_t block);
```





# Dispatch

- ✦ GCD utilise des blocs qui sont dispatchers sur des files.
- ✦ Les files sont gérées par GCD et optimisées en fonction de la charge et du nombre de coeurs du CPU

```
DispatchQueue.global(qos: .background).async {  
  
}
```

# Exemple

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), ^(void) {  
    //do something in background  
    dispatch_async(dispatch_get_main_queue(), ^(void) {  
        //do something in the foreground  
    });  
});
```



# Exemple

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), {  
    //do something in background  
  
    dispatch_async(dispatch_get_main_queue(), {  
        //do something in the foreground  
    });  
});
```



# Exemple

```
DispatchQueue.global(qos: .background).async {  
    //Do something in background  
  
    DispatchQueue.main.async {  
        //Go back in the foreground  
    }  
}
```

# Accéder au réseau

# Généralités

- ✦ (NS)URLSession
  - ✦ Plus moderne et efficace, surtout pour les tâches de téléchargement ou d'upload
  - ✦ Utilise des (NS)URLSessionTask (data, upload & download)



# (NS)URLSession

```
let url = URL(string: "http://monURL/")!
let task = URLSession.shared.dataTask(with: url) { (data, urlResponse, error) -> Void in

    //Check for errors and handle data
}

task.resume()
```

# Parsing JSON



# Généralités

- ✦ JsonSerializer

- ✦ Permet de créer un objet à partir d'un JSON et inversement
- ✦ Nécessite de caster à grand renfort de as?

```
class func data(withJSONObject obj: Any, options opt: JSONSerialization.WritingOptions) throws -> Data
class func jsonObject(with data: Data, options opt: JSONSerialization.ReadingOptions) throws -> Any
```



# JSONSerializer

```
if let statusesArray = try? JSONSerialization.jsonObject(with: data, options: .allowFragments) as?
[[String: AnyObject]],
    let user = statusesArray?[0]["user"] as? [String: AnyObject],
    let username = user["name"] as? String {
    // Finally we got the username
}
```

# Animer son interface

# Généralités

- ✦ Core Animation anime des vues
- ✦ On modifie des propriétés et Core Animation générer l'animation
- ✦ Possibilité de générer très simplement quelques animations, ou de faire des choses plus complexes
  - ✦ Méthode de classe de UIView pour créer un bloc simple d'animation
  - ✦ Voir la classe CABasicAnimation pour plus de contrôle



# Exemple

```
UIView.animate(withDuration: 0.2, animations: {  
    self.button.alpha = 0.5  
    self.view.backgroundColor = UIColor.red  
})
```

# Outils tiers utiles

# Gestionnaires de dépendances

- ✦ CocoaPods
  - ✦ Basé sur un répertoire centralisé
  - ✦ Le plus ancien
- ✦ Carthage
  - ✦ Gestionnaire décentralisé
  - ✦ Plus simple
- ✦ Swift Package Manager
  - ✦ Inclus avec Swift 3.0
  - ✦ Pas encore finalisé, pas de support d'iOS pour le moment

# Alamofire

- ✦ Elegant networking in Swift
  - ✦ Ajoute une surcouche et des fonctions sur URLSession
  - ✦ Création de requêtes, gestion des réponses, validation, etc.
- ✦ Cocoapods : `pod 'Alamofire'`
- ✦ Carthage : `github "Alamofire/Alamofire"`





# Alamofire

```
request(.GET, "https://myapi.com/endpoint").responseJSON { (response) in
    print(response.request) // original URL request
    print(response.response) // HTTP URL response
    print(response.data) // server data
    print(response.result) // result of response serialization

    if let JSON = response.result.value {
        print("JSON: \(JSON)")
    }
}
```

# SwiftyJSON

- ✦ Simplifie l'utilisation du JSON en Swift
  - ✦ La rigueur de Swift sur les types et les optionnels "complexifient" le parsing
  - ✦ SwiftyJSON rend plus agréable l'usage de fichiers JSON
- ✦ Cocoapods : `pod 'SwiftyJSON'`
- ✦ Carthage : `github "SwiftyJSON/SwiftyJSON"`

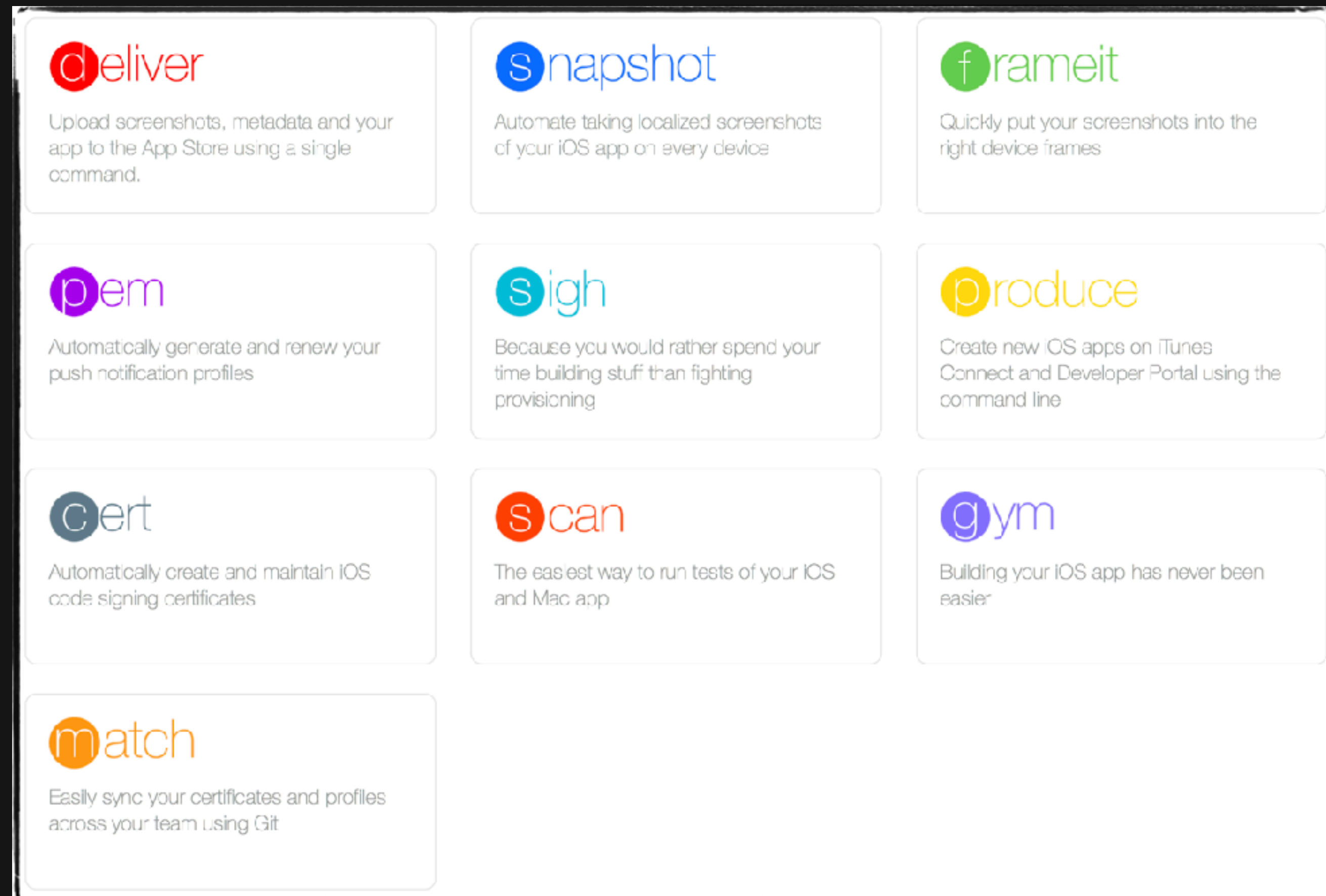


# SwiftyJSON

```
let json = JSON(data: data)
if let userName = json[0]["user"]["name"].string {
    //Now you got your value
}
```

# Fastlane

- ✦ Collection d'outils d'automatisation pour développeurs



# Pour aller plus loin...

- ✦ CocoaPods : Gestionnaire de dépendances centralisé
- ✦ Carthage : Gestionnaire de dépendances décentralisé
- ✦ CocoaControls : répertoire de composants UI

