Business
Services

Développement d'applications iOS

Découverte de Cocoa Touch

Développement d'applications iOS

1. Structure d'un nouveau projet
2. Construction d'IHM
3. Multithreading
4. Core Data

Développement d'applications iOS

> Structure d'un nouveau projet

- main.m
- prefix.pch
- InfoPlist.string
- AppDelegate

Fichiers projet : main.m

- Point d'entrée de l'application
- Très rarement des modifications à apporter
- Petite remarque : en l'ouvrant, on voit pourquoi le main thread a un thread pool initialisé par défaut
- Finalement pas très intéressant…

Fichiers projet : prefix.pch

> Permet de faire des imports qui seront valable dans tous les fichiers de l'application

Fichiers projet : InfoPlist.strings

> Fichier(s) d'internationalisation

- Si on gère plusieurs langues, il y a autant de fichier

Dans InfoPlist.strings (English) :

```
"label.welcome" = "Hello!"
```

Dans InfoPlist.strings (French) :

```
"label.welcome" = "Bonjour !"
```

Dans le code :

```
NSString *label = NSLocalizedString(@"label.welcome",
    @"");
```

Fichiers projet : AppDelegate

> Classe qui reçoit les principaux évènements du cycle de vie de l'application
> Interactions avec l'OS (notifications)

```
- application:didFinishLaunchingWithOptions: // lancement
- applicationWillResignActive: // ex: appel téléphonique
- applicationDidBecomeActive:
- applicationDidEnterBackground: // appui sur 'home'
- applicationWillEnterForeground: // retour
- applicationWillTerminate: // fin d'execution
- applicationDidFinishLaunching: // avant iOS 3
```

Fichiers projet : AppDelegate

> Détails de la méthode appelée au lancement :

```
- (BOOL)application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary
   *)launchOptions
{
   self.window = [[[UIWindow alloc]
   initWithFrame:[[UIScreen mainScreen] bounds]]
   autorelease];
   // Override point for customization after application
   launch.
   self.window.backgroundColor = [UIColor whiteColor];
   [self.window makeKeyAndVisible];
   return YES;
}
```

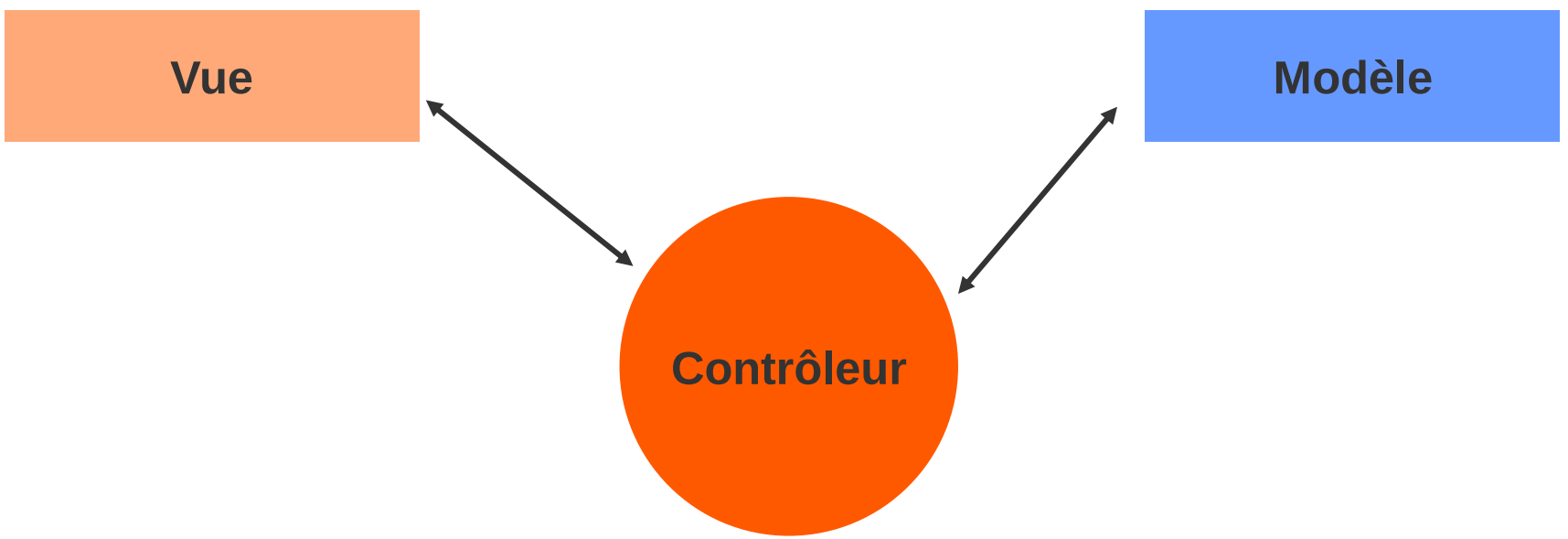
Développement d'applications iOS

> Construction d'IHM : Framework UIKit

- Le pattern MVC
- UIView
- UIViewController
- Organiser la navigation
- Focus sur UITableView

IHM : Le pattern MVC

> Séparation de la logique métier et de la présentation
> Pas d'interaction vue – modèle directe : orchestration par un contrôleur



La vue : UIView

> Classe « racine » des vues dans Cocoa
> Une objet UIView est :
  - un rectangle affichable (et animable)
  - apte à recevoir des évènements utilisateur (touch, drag, etc…)
  - capable de gérer des contenus (sous-vues)

La vue : UIView

> Tous les composants graphiques sont des UIView :
  - UIButton
  - UILabel
  - UIImageView
  - UITextView
  - UIWebView
  - UITableView
  - UIScrollView
  - Etc…

Le contrôleur : UIViewController

> Permet d'interagir avec les composants de la vue d'un côté et le modèle de l'autre
> Il reçoit les évènements de la vue via de méthodes de callback
> Principe lorsqu'on créé un nouvel écran d'IHM :
  - on sous classe UIViewController pour gérer nos composants graphique
  - on surcharge les méthodes qui nous intéressent

Le contrôleur : UIViewController

> Méthodes du cycle de vie de la vue :

- viewDidLoad : vue chargée : appelée une seule fois
- viewWillAppear : la vue va apparaître : appelée N fois
- viewDidAppear : la vue est affichée : appelée N fois
- viewWillDisappear : la vue va disparaître : appelée N fois
- viewDidDisapear : la vue a disparue : appelée N fois
- viewDidUnload : la vue est déchargée, le dealloc est proche

Le contrôleur : UIViewController

> Cas pratique, créer une vue avec :
  - deux UILabel
  - deux UITextField
  - un UIButton
> Implémentation 100% code
> Implémentation avec l'utilisation de Interface Builder

Le contrôleur : UIViewController

> Implémentation 100% code

- La création d'objets graphiques doit se faire dans le viewDidLoad
- Il faut setter les dimensions et le placement des objets avec la propriété frame (penser à la macro CGRectMake)

Le contrôleur : UIViewController

> Implémentation avec Interface Builder

- Les objets graphiques liés au contrôleur doivent être précisés avec la directive **`IBOutlet`**
- Les actions doivent être précisées avec la directive **`IBAction`** comme type de retour
- Vérifier que le File's owner du xib est bien le view controller attendu !

IHM : La navigation

> Une application est en général composée de plusieurs écrans, donc avec plusieurs view controllers

> Pour gérer la transition entre chacun des écrans, UIKit propose en standard des ViewControllers dédiés à cette tâche :

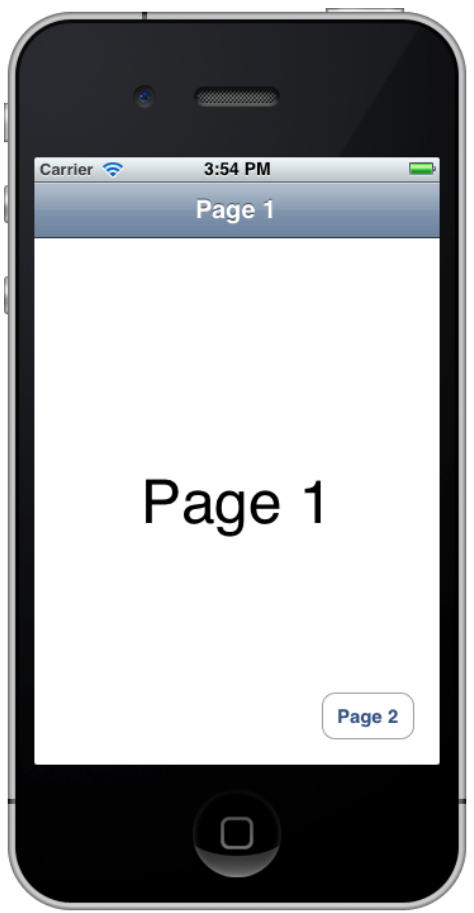
- UINavigationController
- UITabBarController
- UISplitViewController (iPad seulement)

UINavigationController

- Le navigation controller gère une PILE de view controllers.
- Pour passer à l'écran suivant, on « push » un view controller qui est retenu par le navigation controller
- Pour revenir à l'écran précédent, on « pop » le view controller courant. Il est alors relâché par le navigation controller.
- Quand on ajoute un élément à la pile, les précédents ne sont pas désalloués !

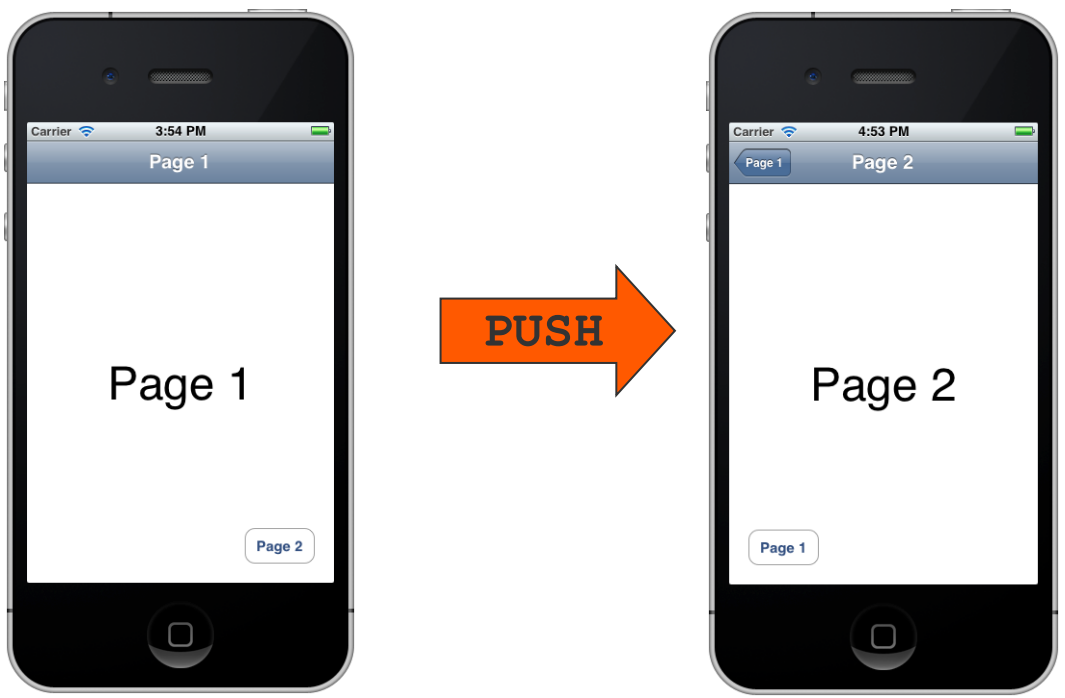
UINavigationController : création

```
UINavigationController
    *navigationController =
    [[UINavigationController alloc]
    initWithRootViewController:page1Vc];
self.window.rootViewController =
    navigationController;
     [page1Vc release];
```



UINavigationController : push

```
Page2 *page2Vc = [[Page2 alloc] init];
[self.navigationController pushViewController:page2Vc
    animated:YES];
[page2Vc release];
```

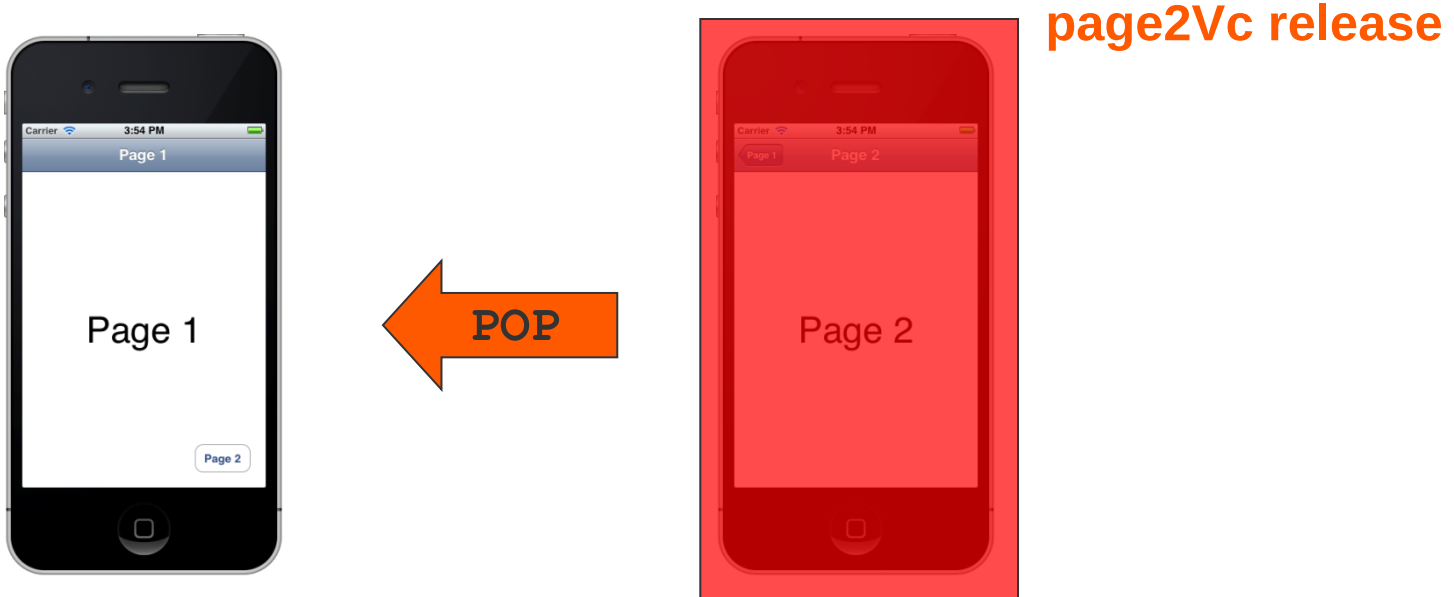


page2Vc retain

UINavigationController : pop

> Le bouton « back » de la navigation bar fait un pop
> Programmatiquement :

```
[self.navigationController popViewControllerAnimated:YES];
```

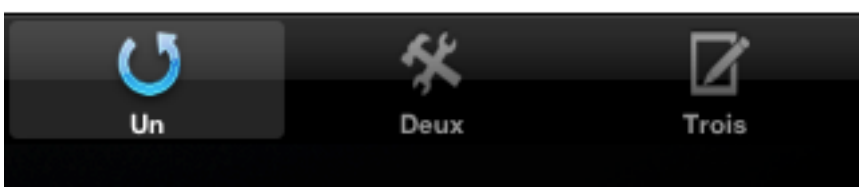


page2Vc release

UITabBarController

> Le tab bar controller gère une liste de view controllers

> Chaque vue est accessible via un bouton dans la tab bar



UITabBarController : création

```
Tab1 *tab1 = [[Tab1 alloc] init];
UIImage* im1 = [UIImage imageNamed:@"refresh"];
UITabBarItem *tabBarItem1 = [[UITabBarItem alloc]
   initWithTitle:@"Un" image:im1 tag:1];
tab1.tabBarItem = tabBarItem1;
[tabBarItem1 release];
/* ... création des autres view controllers ... */
NSArray *array = [NSArray arrayWithObjects:tab1, tab2,
   tab3, nil];
UITabBarController *tabBarController =
   [[UITabBarController alloc] init];
tabBarController.viewControllers = array;
```

UITabBarController

UITableView

> L'objet UITableView permet de gérer une liste d'éléments dans une table

> La liste peut être segmentée en sections

UITableView : principe

- On ne doit pas créer autant de cellules qu'il y'a d'items dans la table, mais seulement celles qui sont visibles (perfs)
- Les cellules créées sont recyclées lorsque l'utilisateurs fait défiler la liste
- Avantage de ce fonctionnement : on peut avoir des très longues listes et préserver un affichage fluide (si on fait pas n'importe quoi non plus)

UITableView : mise en place

> Pour fonctionner, le table view à besoin de deux composants
  - Une data source : fourni les cellules et toutes les informations à afficher
  - Un delegate : permet d'interagir avec le table view
    - Le delegate est facultatif mais on peut rarement s'en passer

UITableView : data source

> La data source doit adopter le protocole UITableViewDataSource

> Deux méthodes obligatoires :

- La première fournit les cellules à afficher

```
- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath
```

- La seconde donne le nombre de cellule dans une section données

```
- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section
```

Par défaut il n'y a qu'une section !

UITableView : data source

> Exemple d'implémentation avec comme hypothèse que les données sont de simples chaines de caractères dans un tableau.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
        cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    NSString *cellIdentifier = @"myCellType";
    UITableViewCell *cell = [tableView
        dequeueReusableCellWithIdentifier:cellIdentifier];

    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                reuseIdentifier:cellIdentifier] autorelease];
    }

    NSString *str = [self.datasource objectAtIndex:indexPath.row];
    cell.textLabel.text = str;
    cell.imageView.image = [UIImage imageNamed:@"logo_orange"];

    return cell;
}
```

UITableView : data source

```
- (NSInteger)tableView:(UITableView *)tableView
      numberOfRowsInSection:(NSInteger)section {

    return [self.datasource count];
}
```

UITableView : delegate

> Le delegate doit adopter le protocole UITableViewDelegate

> Aucune méthodes obligatoire : on implémente en fonction des besoins :

- Sélection utilisateurs
- Hauteur des cellules
- Customisation des headers/footers des sections
- Edition
- Etc…

UITableView : delegate

> Exemple : sélection utilisateur

```
- (void)tableView:(UITableView *)tableView
      didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    NSString *selected = [self.datasource
       objectAtIndex:indexPath.row];
    NSLog(@"selection de %@", selected);
    Page1 *p1Vc = [[Page1 alloc] init];
    [self.navigationController pushViewController:p1Vc
       animated:YES];
    [p1Vc release];
}
```

Développement d'applications iOS

> Multithreading

- NSThread
- NSObject
- NSOperationQueue
- GCD
- Synchronisation

NSThread

> La classe NSThread permet de contrôler l'exécution d'un thread

> Création et lancement :

```
NSThread *t = [[NSThread alloc] initWithTarget:self
   selector:@selector(count) object:nil];
[t start]; // lancement
```

> Ne pas oublier l'autorelease pool !

NSThread

- NSThread propose également des méthodes pour contrôler son exécution :
  - `- isExecuting`
  - `- isFinished`
  - `- isCancelled`
  - `+ exit`
  - `- cancel`
- Priorisation
  - `+ threadPriority`
  - `- threadPriority`
  - `+ setThreadPriority:`
  - `- setThreadPriority:`

NSObject

> La classe NSObject propose des méthodes qui permettent de lancer un thread

> Ce sont des méthodes équivalentes à ce que fait NSThread, voir nécessite un objet NSThread

- `performSelectorOnMainThread:withObject:waitUntilDone:`
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:`
- `performSelector:onThread:withObject:waitUntilDone:`
- `performSelector:onThread:withObject:waitUntilDone:modes:`
- `performSelectorInBackground:withObject:`

NSObject

> Petit aparté : sur iOS seul le main thread a la main sur UIKit
> Si une tâche sur un thread secondaire doit provoquer une mise à jour de l'affichage, les méthodes suivantes sont bien pratiques :

- `performSelectorOnMainThread:withObject:waitUntilDone:`
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:`

On peut rappeler le main thread depuis un thread secondaire

NSOperationQueue

> Permet de créer une file de tâches qui seront exécuter dans des threads secondaires
> Nombre d'opérations exécutées en parallèle configurable
> Permet de contrôler les tâches :
  - Suspendre
  - Annuler

NSOperationQueue : ajouter une tâche

> Deux manières de procéder :
  - Ajouter un block (cf blocks) avec
  - `(void)addOperationWithBlock:(void (^)(void))block`
  - Ajouter un objet NSOperation avec
  - `(void)addOperation:(NSOperation *)operation`

NSOperationQueue : ajouter une tâche

> Avec un block

```
NSOperationQueue *queue = [[NSOperationQueue alloc] init];

queue.maxConcurrentOperationCount = 5;

/* ... */
[queue addOperationWithBlock:^{
    for (int i = 0; i < 20; i++) {
        NSLog(@"%d", i);
    }
}];
```

NSOperationQueue : ajouter une tâche

> Avec un objet NSOperation

1. Sous classer la classe NSOperation
2. Surcharger la méthode **`main`**
3. Implémenter le « code utile » dans la méthode main

Synchronisation

> Deux méthodes :
  - La directive **`@synchronized`**
  - La classe NSLock
> Avec @synchronized, utilisation simple :
  - On isole le bloc de code qu'on souhaite rendre atomique, exemple :

```
@synchronized (self) { // verrou sur self
   /* Section critique */
}
```

Synchronisation

> Avec NSLock
  - Principe du mutex, pour accéder à une section critique on doit acquérir un verrou, une instance de NSLock
> Exemple :

```
NSLock *lock = [[NSLock alloc] init];
/* ... */
[lock lock]; // thread bloqué ici si verrou déjà pris
/* Section critique */
[lock unlock]; // déverrouille
```

Grand Central Dispatch (GCD)

- GCD est apparu avec iOS 4
- Vise à améliorer la gestion de la concurrence
- Optimisation pour les processeurs multi-cœurs

Grand Central Dispatch (GCD)

- GCD est apparu avec iOS 4
- Optimisation pour les processeurs multi-cœurs
- Trois types de queues :
  - Main queue (sur le main thread)
  - Private dispatch queue (exécution en série sur un thread secondaire)
  - Global dispatch queue (exécution en parallèle par des threads secondaires)
- Deux types d'exécutions : synchrone ou asynchrone

GCD : Main Queue

- S'exécute sur le main thread
- Utile pour des mise à jour de l'IHM par exemple
- Obtention :

```
dispatch_queue_t main_queue = dispatch_get_main_queue();
```

GCD : Private Dispatch Queue

- Queue locale au thread courant
- Les opérations sont exécutées de manière séquentielle
- Obtention :

```
dispatch_queue_t queue =
    dispatch_queue_create(@"com.orange.privateQueue",
    NULL); // second param toujours NULL
```

- Ne pas oublier de libérer la queue !

```
dispatch_release(queue);
```

GCD : Global Dispatch Queue

- Queues globales à l'application
- Une queue par niveau de priorité
- Les opérations sont exécutées en parallèle
- Obtention :

```
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAU
   LT, NULL); // dernier param toujours NULL
```

GCD : Global Dispatch Queue

> Les niveaux de priorité
  - `DISPATCH_QUEUE_PRIORITY_HIGH`
  - `DISPATCH_QUEUE_PRIORITY_DEFAULT`
  - `DISPATCH_QUEUE_PRIORITY_LOW`
  - `DISPATCH_QUEUE_PRIORITY_BACKGROUND // iOS 5`

> Le dernier niveau vise à minimiser l'impact sur le système

GCD : Utilisation

> Appel asynchrone

```
dispatch_queue_t queue = /* selon type de queue*/
dispatch_async(queue, ^{
    /* code block */
});
```

> Appel synchrone

```
dispatch_sync(queue, ^{
    /* code block */
});
```

Développement d'applications iOS

> Core Data

- Présentation
- Pile Core Data
- Modélisation
- Requêtes
- Insertion / mise à jour / suppression / sauvegarde

Core Data : Présentation

> Core Data c'est :
  - Un framework de persistance des données + mapping relationnel objet
  - Simple à mettre en place
  - Conçu pour les applications desktop (et mobile), pas les serveurs
> En conséquence :
  - C'est relativement simple à mettre en place
  - Ça semble parfois limité si on a connu autre chose (Hibernate and co)

Core Data : La pile

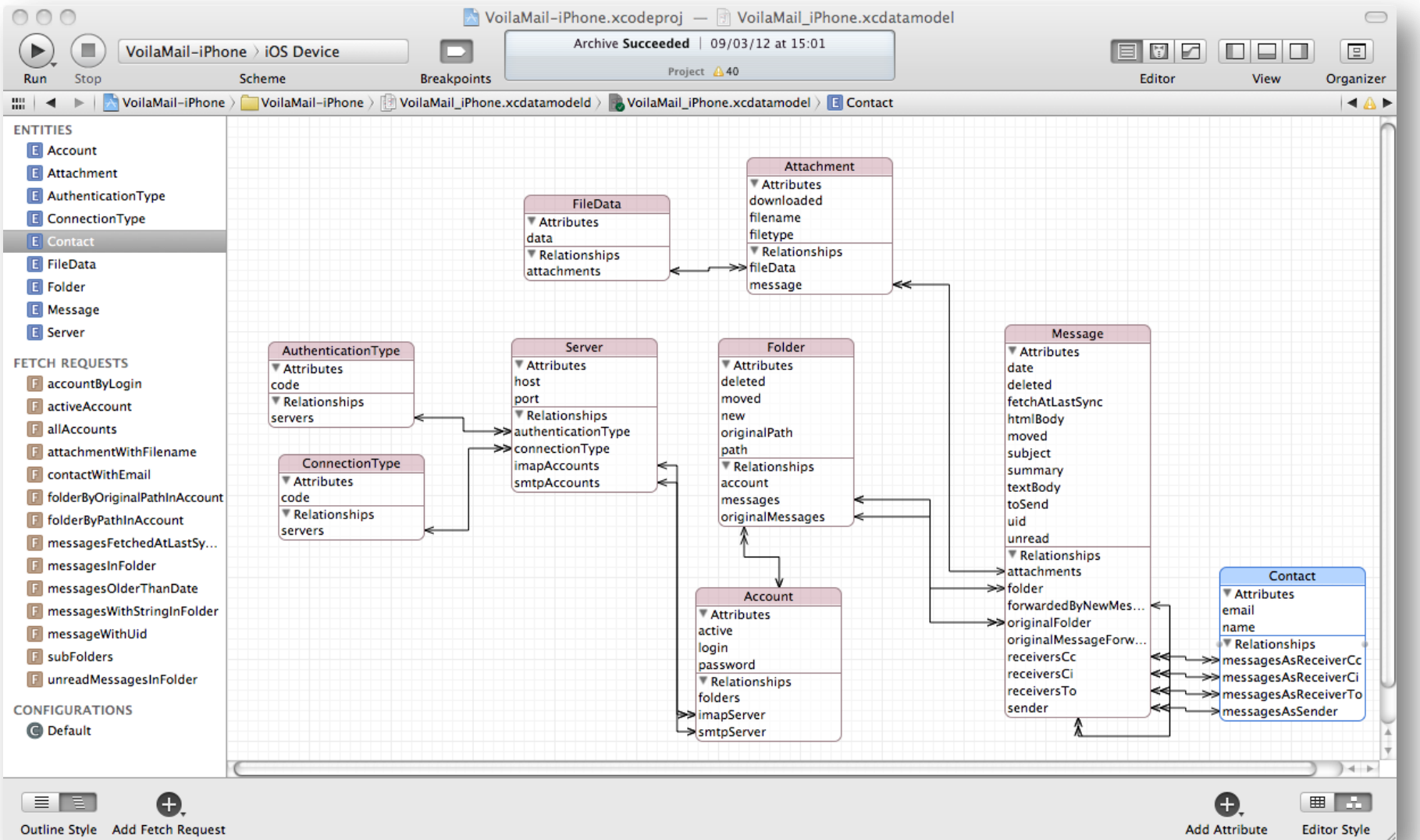
> La pile Core Data est composée de 3 objets principaux :

- Le Persistent Store Coordinator
  - Couche d'abstraction entre l'API et la solution de persistance choisie (XML, SQLLite, mémoire)
- Le Managed Object Model
  - Représente la structure de nos données « à vide » et en statique
- Le Managed Object Context
  - Représente nos données en dynamique, c-à-d nos objets qui peuplent le modèle

Core Data : Modélisation

> Editeur inclus dans Xcode
  - Modélisation des données
  - Génération du code des classes des entités
  - (perfectible)

Core Data : Modélisation



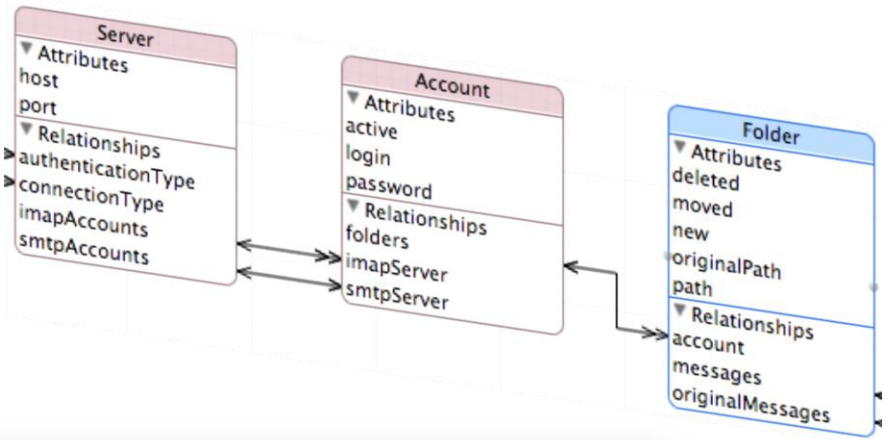
Core Data : Modélisation

> Une entité c'est :

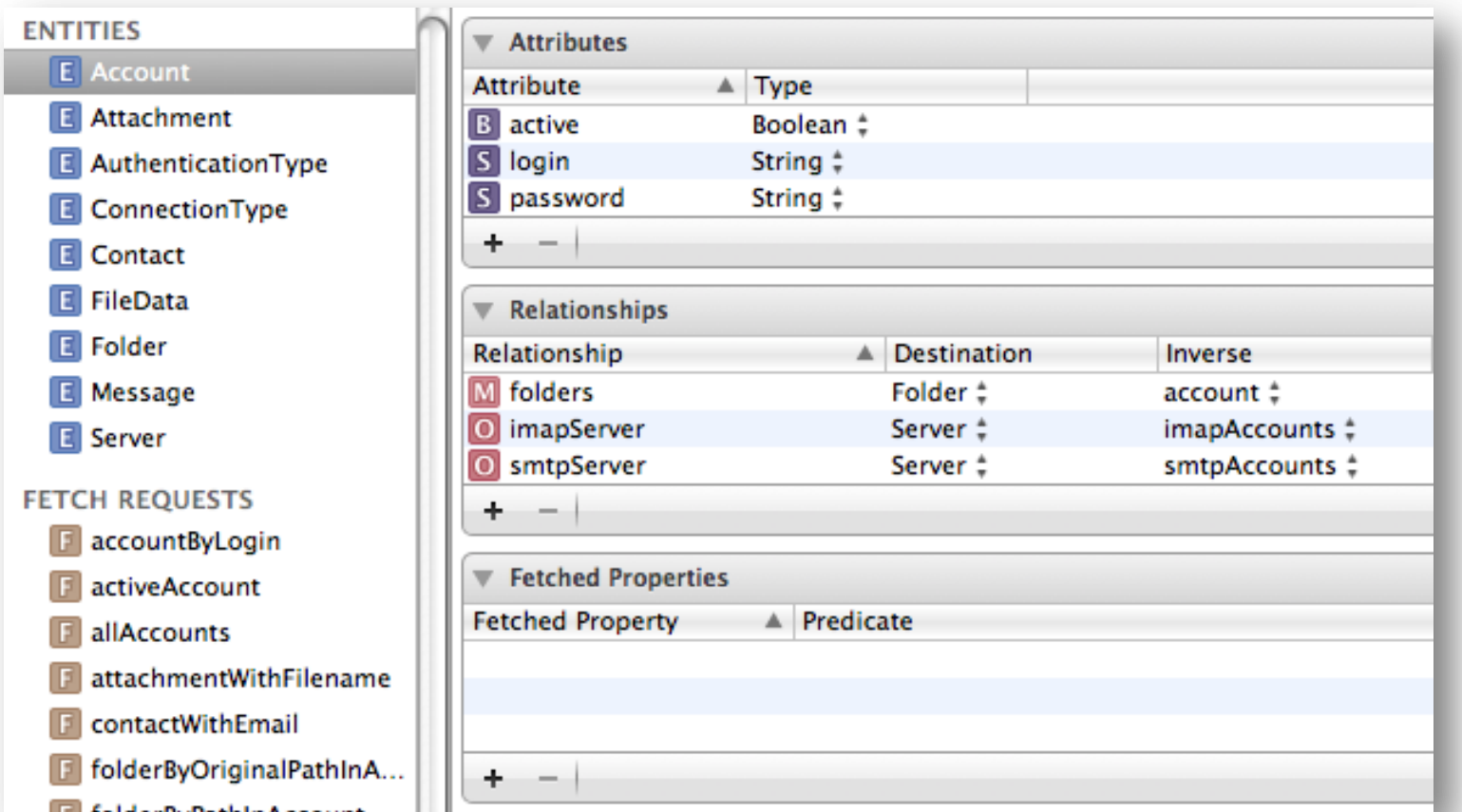
- Des propriétés (données)
- Des relations avec d'autres entités. Il en existe 2 types :
  - Relationship
  - Fetched property (peu utilisée)

Chaque entité donnera une classes, chaque relation une propriété.

Core Data : Modélisation



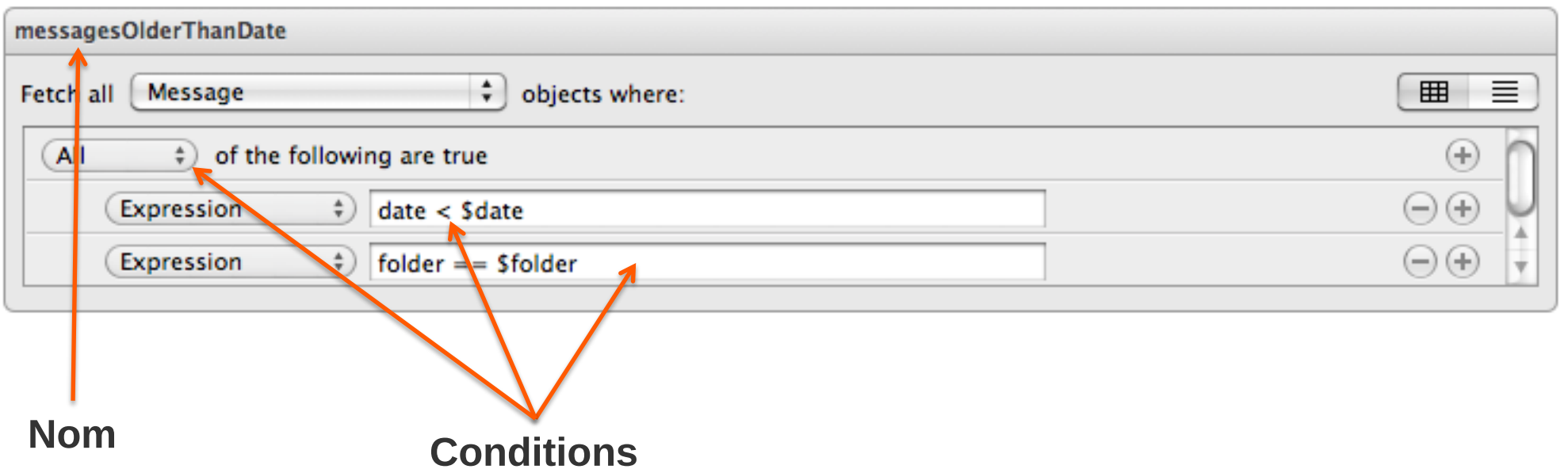
> Exemple



Business Services orange

Core Data : Requête

> Première étape, créer une fetched request dans Xcode



Core Data : Requête

> Seconde étape, appeler la fetch request dans le code

```
NSDictionary *vars = [NSDictionary dictionaryWithObjects:
                    [NSArray arrayWithObjects:folder, date, nil]
            forKeys:[NSArray arrayWithObjects:@"folder", @"date", nil]];
NSFetchRequest *request =[self.managedObjectModel
     fetchRequestFromTemplateWithName:@"messagesOlderThanDate"
     substitutionVariables:vars];
NSError *error;
NSArray *messages = [self.managedObjectContext
                        executeFetchRequest:request error:&error];
```

Core Data : Insertion

> Créer et initialiser une instance de l'entité

```
OBSFolder *folder = [NSEntityDescription
                    insertNewObjectForEntityForName:@"Folder"
                    inManagedObjectContext:self.managedObjectContext];
folder.path = path;
folder.account = account;
```

Core Data : Sauvegarde

```
NSError *error;
if (![self.managedObjectContext save:&error]) {
    NSLog(@"Delete attachment. Cannot save managed object context,
    %@", [error localizedDescription]);
}
```