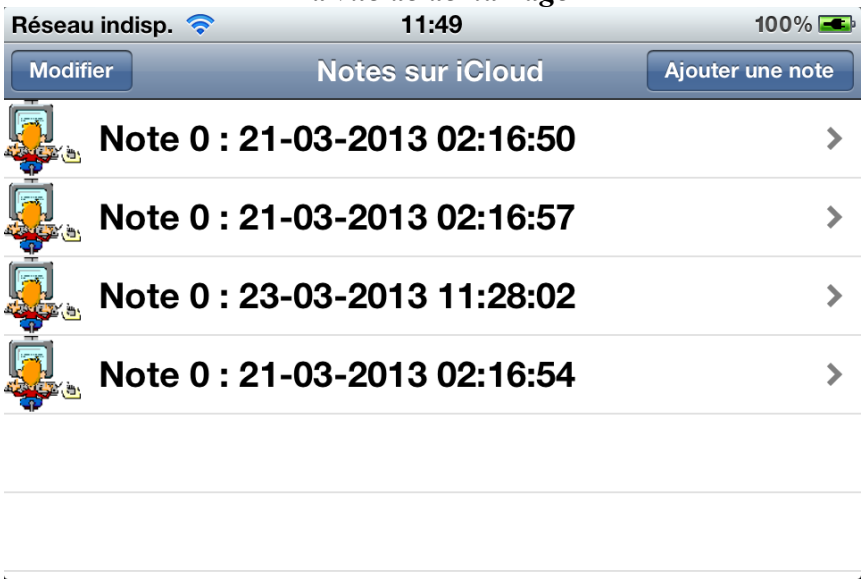
Manipulation 7 : Stockage iCloud.

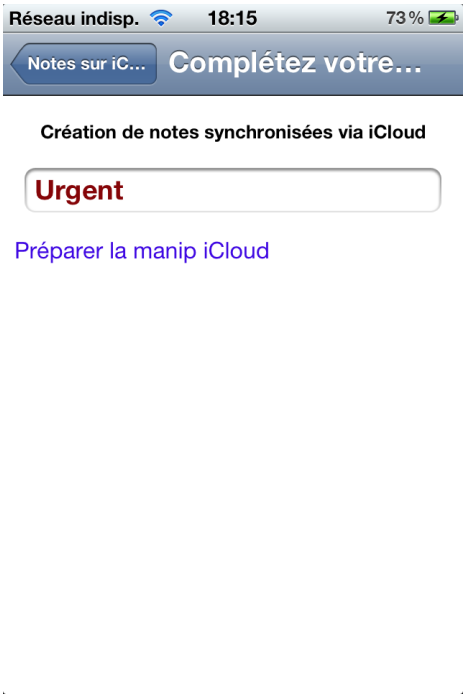
Objectif :

Cette manipulation a pour but d'utiliser le « cloud » pour sauvegarder des documents de type « note » gérés par une application iPx. Ces simples notes seront constituées d'un sujet et d'un message. Elles seront listées dans la vue de démarrage de l'application permettant l'ajout ou la suppression de notes et éditables dans une vue secondaire dite « vue de détail ».

La vue de démarrage



La vue d'édition d'une note



Projet de départ

Un projet de départ vous est fourni afin de concentrer l'activité de la manipulation sur la gestion des documents **iCloud**. Dans le projet initial les parties décrites ci-dessous ont déjà été conçues, pour ceux qui désireraient partir de zéro la marche à suivre pour aboutir à ce point de « départ avancé » est fournie en annexe.

Travail (essentiellement lié au design des vues) déjà réalisé :

- ✓ l'ajout des icônes et images de chargement de l'application,
- ✓ la conception graphique du **storyboard** avec une vue simple (**UIViewController**) et une vue liste (**UITableViewController**),
- ✓ la conception du « **Navigation Controller** » permettant la navigation entre les deux vues,
- ✓ l'association des classes contrôleurs aux vues du **storyboard**,
- ✓ la conception graphique de la cellule élémentaire, modèle de la liste,
- ✓ la déclaration des attributs et propriétés associés aux contrôles graphiques,
- ✓ la déclaration de la méthode (**IBAction**) déclenchée lors du clic sur le bouton « **Ajouter une note** ».

Préparation du projet pour les accès iCloud, le « profil de provisionnement »

Par souçi de simplicité (pour le formateur…) tous les projets auront le même identificateur « **App ID** » (**Application Manip 7 iCloud**). Celui-ci a normalement déjà dû être créé sur le portail de provisionnement.

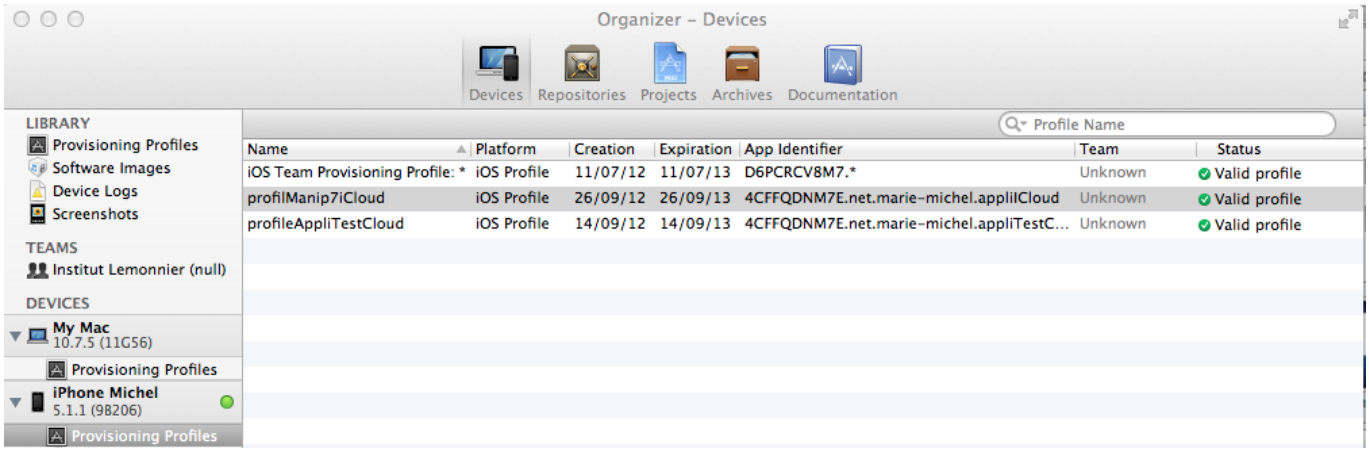
Vous pouvez ensuite éditer en mode « modifications » le profil de provisionnement (**profilManip7iCloud**) associé au précédent « **App ID** » et le modifier si nécessaire (ajout de votre iDevice par exemple s'il n'est pas autorisé pour ce profil).

Il est cependant conseillé d'aller sur le portail d'Apple afin d'éditer l'identificateur de votre application et de vérifier qu'il est « iCloud compatible ».

https://developer.apple.com/ios/manage/overview/index.action

Une fois que le profil est vérifié par chaque groupe surtout ne pas renouveler le profil de provisionnement qui est commun à tous les groupes car cela obligerait chacun à détruire les profils déjà installés pour recharger le profil renouvelé…

Il reste à télécharger et à installer le profil de provisionnement sur votre poste de développement, ce qui se fait simplement en double cliquant sur le fichier de profil une fois téléchargé.
Vous pouvez ensuite constater son installation avec l'outil « **organizer** » comme ci-dessous :



La conception de l'application iCloud

Partie 1 : Configurer l'accès iCloud.

1. Ouvrir le projet fourni et afficher la fenêtre « **TARGETS->summary** » puis descendre à la rubrique des autorisations iCloud « **entitlement** ». Configurer les autorisations iCloud suivantes :
   - ✓ fichier d'autorisations = **appliICloud**,
   - ✓ iCloud validé,
   - ✓ conteneur d'ubiquité = « **MM.appliICloud** »,
   - ✓ groupe des clés de sécurité (**App ID**) = « **net.marie-michel.appliICloud** ».
2. Compléter la classe délégué d'application, afin qu'elle vérifie l'accès iCloud au démarrage de l'application et qu'elle signale l'éventuel échec de l'accès iCloud dans une fenêtre **UIAlertView**. Ce test servira de fait à créer les conteneurs ubiquité pour l'application lors de son premier démarrage.
3. Vérifier votre accès iCloud après avoir configuré l'accès iCloud du périphérique en utilisant l'identifiant Apple suivant :
   - ✓ Utilisateur : michel.marie@institutlemonnier.fr,
   - ✓ Mot de passe : **Password1234**.

Partie 2 : La classe « Note » sous-classe UIDocument

Cette partie va consister à concevoir une classe du type **UIDocument** afin de modéliser les documents de type « **Note** » constitués d'un sujet et d'un message.

1. Ajouter à votre projet un nouveau fichier du type Objective-C, la classe sera nommée « **Note** » et héritera de la classe de gestion des documents **UIDocument**.
2. Ajouter à cette classe les attributs et propriétés associées de type **NSString*** nommés **noteMessage** et **noteSujet**.

   Le document note aura donc pour données les informations issues de ces deux attributs. Ces données seront écrites dans le fichier document en utilisant le format JSON avec simplement :
   - ✓ une clé « **Sujet** » pour l'accès au sujet de la note,
   - ✓ une clé « **Message** » pour le message de la note.

   Ci-dessous un exemple du contenu d'un fichier note au format JSON correspondant :

   ```
   {
    "Message" : "http://www.fr.lastminute.com/site/week-end-pas-cher",
    "Sujet" : "Préparer son Week-end"
   }
   ```

3. Surcharger la méthode de lecture des données du document « note » **loadFromContents:**, méthode de lecture des données du fichier document dans l'objet note associé. Les données issues du fichier sont passées par l'intermédiaire du premier paramètre de la méthode. Cette méthode devra :
   - ✓ vérifier que le paramètre « **contents** » contient des données,
     i. Si c'est le cas extraire ces données pour les recopier dans les attributs de la classe (**noteSujet** et **noteMessage**),
     ii. Sinon écrire une valeur par défaut dans les attributs (vous pouvez d'ores et déjà penser à localiser votre application).
   - ✓ retourner YES.

   Pour la lecture des données au format JSON vous pouvez utiliser conformément à l'exemple du document de cours :
   - ✓ la classe **NSJSONSerialization** méthode **dataWithJSONObject**,
   - ✓ la méthode **valueForKey**.

4. Surcharger la méthode d'écriture des données du document « note » **contentsForType:**, méthode appelée lors de la sauvegarde du document depuis l'objet note associé. Les données à écrire doivent-être retournées par la méthode (utilisation de la classe **NSData** ou **NSFileWrapper**). Cette méthode devra :
   - ✓ vérifier que l'attribut **noteMessage** contient un message et si ce n'est pas le cas écrire une valeur par défaut dans les attributs (les mêmes valeurs par défaut qu'à la question précédente),
   - ✓ créer un dictionnaire avec les clés « **Sujet** » et « **Message** » et les valeurs associées issues des attributs correspondants,
   - ✓ recopier ce dictionnaire dans un objet de la classe **NSData** et retourner celui-ci.

   Pour l'écriture des données au format JSON vous pouvez utiliser :
   - ✓ la classe **NSDictionnary** méthode **dictionnaryWithObjectsAndKeys**,
   - ✓ la classe **NSJSONSerialization** méthode **JSONObjectWithData**.

Surcharger les propriétés en écriture pour les attributs **noteSujet** et **noteMessage** afin d'activer l'auto-sauvegarde des documents par enregistrement auprès de l'objet de type **NSUndoManager** membre du document. Si vous avez respecté la syntaxe proposée en 5) la signature de ces propriétés doit être la suivante :

```
- (void)setNoteMessage:(NSString *)newNoteMessage
- (void)setNoteSujet:(NSString *)newNoteSujet
```

Dans notre applications nous souhaitons utiliser un délégué pour être prévenu du chargement d'un document iCloud (lors d'une modification de celui-ci par exemple), ceci nous sera utile pour mettre à jour la vue de détail et afficher le contenu du document note iCloud, sujet et message.

La première étape pour ajouter la prise en compte d'un délégué consiste à définir un protocole avec les méthodes que cet objet doit implémenter.

5. Définir le délégué pour la classe **Note**, pour ceci :
   - ✓ dans le fichier **Note.h** déclarer un nouveau protocole nommé **NoteDelegate** avec une méthode optionnelle **–(void)noteContentsDidChange:(Note*)noteModifiee**,
   - ✓ ajouter la déclaration du protocole avant la définition de la classe,
   - ✓ ajouter à la classe **Note** une propriété nommée **delegate** avec référence faible (weak) pour ce protocole.

   Le code associé à ce protocole devrait ressembler à ceci :

```
// Note.h
@protocol NoteDelegate;

@interface Note : UIDocument
@property (weak, nonatomic) id<NoteDelegate> delegate;
@end

@protocol NoteDelegate <NSObject>
@optional
- (void)noteContentsDidChange:(Note*)noteModifiee;
@end

// Note.m
@synthesize delegate;
```

6. Ajouter l'appel de la méthode du délégué à la fin de la méthode de chargement du document **Note** : **loadFromContents:**. Il est nécessaire de vérifier que l'objet délégué existe et qu'il prend en charge la méthode associée (via **respondsToSelector**) avant de déclencher celle-ci.

   Le code associé dans la méthode **loadFromContents:** devrait ressembler à ceci :

```
   if (self.delegate && [self.delegate
respondsToSelector:@selector(noteContentsDidChange:)])
        [self.delegate noteContentsDidChange:self];
```

Partie 3 : La vue de démarrage avec la liste des notes, UITableView.

Cette partie va consister à compléter la vue d'affichage de la liste des notes, gérée par le contrôleur **ViewControllerTableNotes**.

1. Ajouter à cette classe les données membres suivantes :
   - ✓ **notes** de type **NSMutableArray*** pour la gestion de la liste des objets **note**,
   - ✓ **metadataQuery** de type **NSMetadataQuery***, objet de recherche sur les conteneurs ubiquité.
2. Compléter la méthode **viewDidLoad** afin de faire en sorte que :

- ✓ le titre de la vue affiche « **Notes sur iCloud** »,
- ✓ le tableau « **notes** » soit instancié,
- ✓ soit ajouté un bouton d'édition dans la partie gauche de la barre de navigation, celui-ci nous servira plus tard à activer la suppression des notes.

```
self.navigationItem.leftBarButtonItem = self.editButtonItem;
```

Configuration de la « Table View Data Source ».

3. Compléter si nécessaire la méthode **numberOfSectionsInTableView** afin que la liste ne présente qu'une seule section.
4. Compléter la méthode **numberOfRowsInSection** afin qu'elle retourne le nombre d'objets présents dans la liste des objets note.
5. Compléter la méthode **cellForRowAtIndexPath** afin qu'elle configure la cellule courante (indexée par le paramètre **indexPath**) et retourne celle-ci. Pour ceci :
   - ✓ le nom du fichier (sans le chemin ni l'extension) associé à l'objet note courant sera affecté à la partie « **textLabel** » de la cellule,
   - ✓ la cellule devra également présenter un « indicateur de dévoilement » (propriété **accessoryType**), le chevron incliné situé à droite de la cellule invitant à cliquer dessus pour visualiser le contenu de celle-ci.

Codages des requêtes de surveillance des « conteneurs d'ubiquité ».

Pour la surveillance des conteneurs nous allons utiliser l'objet **NSMetadataQuery** déclaré en attribut.

- ✓ Une méthode **getNoteQuery** sera chargée de retourner un objet de type **NSMetadataQuery** initialisé pour filtrer la requête de surveillance sur les fichiers du dossier **Documents** du conteneur dont le nom commence par « **Note x :** » et d'extension « **.mmc** », avec **x** = le numéro de votre groupe de TP, ceci afin que les différents documents des différents groupes qui se retrouveront dans le même conteneur iCloud ne se mélangent pas.
- ✓ Une méthode **configAndStartQueryByBlock** qui sera exécutée au chargement de la vue principale, à la fin de la méthode **viewDidLoad**, elle sera quant à elle chargée d'activer la surveillance dans des blocs :
  - o de la fin de la première phase de collecte des documents, notification **NSMetadataQueryDidFinishGatheringNotification,**
  - o de la mise à jour de documents suite à une modification détectée dans la partie du conteneur ubiquité surveillé, notification **NSMetadataQueryDidUpdateNotification.**

6. Coder la méthode **–(NSMetadataQuery*) getNoteQuery**, celle-ci devra :
   - ✓ instancier un objet **NSMetadataQuery**,
   - ✓ configurer la requête **metadataQuery** pour effectuer la recherche de documents filtrée sur le « conteneur ubiquité », méthode **setSearchScopes**,
   - ✓ fixer l'intervalle entre les notifications de mise à jour en cas de changement à 1s, méthode **setNotificationBatchingInterval,**
   - ✓ définir un alias **PREDICAT** avec la chaîne de filtrage « **%K like 'Note n : *.mmc'** » (Rappel : %K étant un argument de substitution pour une clé alors que %@ l'est pour un objet),
   - ✓ instancier un objet prédicat (classe **NSPredicate**) avec la chaîne de filtrage **PREDICAT** précédemment définie puis associer celui-ci à la requête de recherche,
   - ✓ retourner l'objet **NSMetadataQuery** ainsi initialisé.

7. Coder la méthode **–(void)configAndStartQueryByBlock**, celle-ci devra :
   - ✓ vérifier si la donnée membre **metadataQuery** est non initialisée :
     i. si oui on initialise celle-ci avec la méthode **getNoteQuery**,

ii. si non on est déjà passé par cette méthode, on stoppe la requête de surveillance, méthode **stopQuery**.

- ✓ activer la surveillance de la notification **NSMetadataQueryDidFinishGatheringNotification** dans un bloc, celui-ci devant :
  i. désactiver les mises à jour sur les résultats de la requête,
  ii. initialiser un tableau **NSMutableArray** avec la liste des URL des documents résultats de la requête (« items » résultat de la requête),
  iii. supprimer le contenu du tableau des URL, donnée membre **notesURL**, pour y recopier le tableau précédent,
  iv. activer les mises à jour sur les résultats de la requête,
  v. recharger la liste « Table View », méthode **reloadData**.
- ✓ activer la surveillance de la notification **NSMetadataQueryDidUpdateNotification** dans un bloc, celui-ci devant réaliser exactement le même travail que la notification de fin de première collecte des documents.
- ✓ activer le démarrage de la requête de surveillance.

8. Vous pouvez à présent tester votre application qui devrait afficher dans la « **Table View** » les notes présentes sur le Cloud respectant le format de fichier filtré par votre objet « **metadataQuery** ». Demander éventuellement au formateur qu'il ajoute des fichiers notes au Cloud.

Ajout d'une nouvelle note.

L'ajout d'une nouvelle note devra créer l'URL du nouveau document afin de l'ajouter à la liste des URL puis ajouter la cellule pour cette nouvelle note à la « Table View » et enfin basculer vers la vue de détail de cette nouvelle note. La vue de détail disposera d'un attribut « noteURL » représentatif de l'URL de la note en cours d'édition.

9. Depuis le **storyboard** créer une transition « segue » entre les cellules de la « Table View » et la vue de détail. Configurer la transition avec le style PUSH et lui donner l'identificateur « **versVueDetailNote** ».
10. Dans la classe **ViewController** ajouter un attribut et la propriété associée **noteURL** de type **NSURL***.
11. Ajouter la méthode **prepareForSegue** à la classe **ViewControllerTableNotes**, celle-ci devra préparer le passage de la « Table View » vers la vue de détail lorsque l'on sélectionne une note ou lorsque l'on ajoute une nouvelle note. Cette méthode devra :
    - ✓ Vérifier que la transition en cours correspond bien à la transition vers la vue de détail, et si c'est bien le cas :
      i. récupérer l'objet de type **NSIndexPath**, index de la cellule sélectionnée,
      ii. recopier l'URL du document associé à la cellule active dans l'attribut **noteURL** de la classe **ViewController**.

12. Compléter la méthode **ajoutNote** celle-ci devant ajouter une nouvelle URL d'un futur nouveau document Note à la donnée membre **notesURL**. Le futur document Note sera stocké à la racine du « conteneur ubiquité » et sera nommé selon le format « **Note x : dd-MM-yyyy HH :mm :ss.mmc** ».
    Le formatage de la date et de l'heure de création à insérer dans le nom du document note pourra se faire en utilisant la classe **NSDateFormatter.**
    Une fois l'URL ajoutée au tableau **notesURL** il faudra ajouter une nouvelle cellule à la table pour ce nouveau document et sélectionner cette cellule.
    L'ajout d'une cellule à la table nécessite :
    - ✓ la création d'un objet **NSIndexPath** pour spécifier l'emplacement de la cellule, méthode **indexPathForRow:**,

- ✓ l'ajout de la cellule à l'emplacement spécifié dans la « table View », méthode **insertRowsAtIndexPaths:**.

La sélection de la cellule spécifiée de la « table View » une fois ajoutée se fait en utilisant la méthode **selectRowAtIndexPath:**.

Ensuite il faudra basculer vers l'affichage de la vue de détail à l'aide de la méthode **performSegueWithIdentifier**.
Enfin, et pour finir, il faudra faire en sorte que le bouton « Ajout » soit invalidé pendant l'exécution de cette méthode.

L'exemple ci-dessous propose une façon de traiter cette partie :

```
#define NOTEFORMAT @"Note %@ : %@.mmc"

    self.buttonAdd.enabled = NO;
    // Retrouve l'URL du répertoire conteneur iCloud dans lequel on veut créer le
document
    NSURL *ubiq = [[NSFileManager
defaultManager]URLForUbiquityContainerIdentifier:nil];
    if (ubiq==nil) {
        // Pas d'accès iCloud, on ne fait rien...
        self.buttonAdd.enabled = YES;
        return;
    }
    NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
    [formatter setDateFormat:@"dd-MM-yyyy HH:mm:ss"];
    NSString *fileName = [NSString stringWithFormat: NOTEFORMAT,
                          [formatter stringFromDate:[NSDate date]]];
    // création du chemin "conteneuriCLoud\filename"
    NSURL* newNoteURL =
     [[ubiq URLByAppendingPathComponent:@"Documents"]
URLByAppendingPathComponent:fileName];
    [notesURL addObject:newNoteURL];
    // MAJ Table.
    NSIndexPath* newCellIndexPath = [NSIndexPath
                          indexPathForRow:([notesURL count] - 1) inSection:0];
    [self.tableView insertRowsAtIndexPaths:[NSArray
arrayWithObject:newCellIndexPath]
                          withRowAnimation:UITableViewRowAnimationAutomatic];
    [self.tableView selectRowAtIndexPath:newCellIndexPath animated:YES
                          scrollPosition:UITableViewScrollPositionMiddle];
    // Basculer vers la vue de détail pour édition.
    [self performSegueWithIdentifier:@"versVueDetailNote" sender:nil];
    // Ré-enable the Add button.
    self.buttonAdd.enabled = YES;
```

13. Tester l'ajout des notes et leur visualisation dans la liste. et sur le Cloud.

Partie 4 : La vue de détail d'une note pour son édition.

Affichage d'une note dans une vue de détail pour modifier les valeurs par défaut.

Lors de la création d'une note, le sujet et le message sont affectés par défaut dans la classe Note. La vue « **ViewController** » de l'application devra s'ouvrir lors de la sélection d'une note dans la liste afin de pouvoir modifier le sujet et le message. Le retour à la vue liste des notes activera la mise à jour de la note modifiée et le rechargement des notes dans la « Table View ».

1. Ajouter une donnée membre nommée **note** de type **Note*** à la classe **ViewController.**
2. Compléter la méthode **viewDidLoad** de la vue de détail afin d'ajouter le titre « **Modifiez votre note** » à la vue.

3. Ajouter la méthode **viewWillAppear** de la vue de détail afin :
   - ✓ d'instancier la donnée membre **note** avec l'URL du document sélectionné, récupéré dans l'attribut **noteURL**,
   - ✓ de définir l'objet courant comme délégué pour l'objet **note**,
   - ✓ de vérifier si le document (fichier) d'URL spécifiée existe,
     i. si c'est le cas l'ouvrir pour activer son chargement iCloud, méthode **openWithCompletionHandler**,
     ii. sinon le créer, méthode **saveToURL**.
   - ✓ de recopier le sujet et le message de la note dans les propriétés d'affichage **noteMessage** et **noteSujet** de la vue de détail.

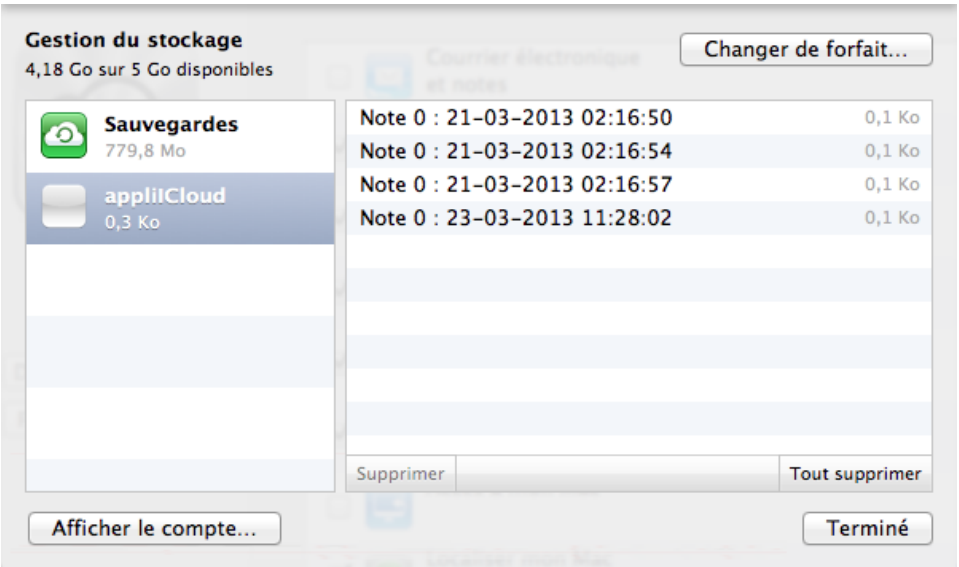
Prise en compte des changements dans la note

4. Ajouter la méthode **viewWillDisappear** de la vue de détail afin :
   - ✓ de recopier les propriétés d'affichage **noteMessage** et **noteSujet** de la vue de détail dans les propriétés **noteSujet** et **noteMessage** de la note,
   - ✓ de fermer le document **note**, méthode **closeWithCompletionHandler**.

5. Coder la méthode délégué du protocole **NoteDelegate** afin qu'elle mette à jour les propriétés d'affichage **noteMessage** et **noteSujet** de la vue de détail.

6. Vérifier le fonctionnement de l'affichage de la vue de détail.

Pour la visualisation des documents sur le Cloud depuis votre Mac ouvrir « **préférences système->iCloud->Gérer->appliICloud** » et vous devriez voir la liste des documents stockés sur iCloud par l'application comme sur la figure suivante :



Vous pouvez aussi le faire depuis l'iPad ou l'iPhone.

Partie 5 : La suppression d'une note.

La suppression d'une cellule de la **TableView** se fait à l'aide de la méthode **commitEditingStyle** qui permet la gestion de la suppression ou de l'insertion d'une cellule. Au sein de cette méthode il faut vérifier que l'appel de celle-ci a pour origine une suppression de cellule et dans ce cas nous allons

récupérer l'URL de l'objet **Note** courant pour le supprimer du conteneur ubiquité puis du tableau **notesURL** liste des notes et enfin supprimer la cellule associée de la **TableView**.

La suppression d'une note se fait à l'aide de la méthode **removeItemAtURL** de la classe **NSFileManager.**
Pour supprimer le document iCloud nous allons utiliser le **GCD** « **Grand Central Dispatch** ».
Il faudra synchroniser la suppression de l'objet **NSURL** du tableau de **notesURL** et de l'objet cellule avec la suppression de l'objet Note, pour ceci nous utiliserons un sémaphore de synchronisation.

7. Dé-commenter la méthode **commitEditingStyle** et ne garder que la partie du traitement liée à la suppression d'une cellule. Compléter la méthode précédente afin qu'elle :
   - ✓ récupère l'URL de l'objet note sélectionné pour la suppression,
   - ✓ désactive les mises à jour de la requête de surveillance des métadonnées,
   - ✓ initialise un sémaphore de synchronisation, méthode **dispatch_semaphore**,
   - ✓ supprime l'objet **note** du conteneur ubiquité au sein d'un thread activé sur la « global queue » de priorité haute, une fois la suppression réalisée, ce thread devra libérer le sémaphore,
   - ✓ attendre la fin du thread de suppression,
   - ✓ supprimer l'objet **NSURL** associé à l'objet note courant du tableau **notesURL**,
   - ✓ supprimer la cellule sélectionnée de la **tableView**,
   - ✓ réactive les mises à jour de la requête de surveillance des métadonnées.

8. Vérifier le fonctionnement de la suppression d'une note depuis l'application ou depuis le gestionnaire iCloud.
9. Vérifier le fonctionnement d'ensemble de l'application en utilisant 2 matériels si possible…

Partie 6 : Le classement des Notes dans l'ordre croissant des noms de fichier.

Les URL des documents affichés dans la table sont stockés dans la donnée membre notesURL, celle-ci étant réinitialisée à chaque modification détectée dans le conteneur d'ubiquité.

Pour afficher les notes dans l'ordre croissant des noms de fichiers nous allons utiliser une des méthodes de la classe **NSMutableArray** prévue pour la réorganisation des tableaux, la méthode **sortUsingSelector**.

Cette méthode permet de ranger les éléments d'un tableau dans l'ordre croissant, la méthode de comparaison entre éléments devant être spécifiée lors de l'appel de la méthode.

La méthode de comparaison est appelée pour chaque objet du tableau et reçoit pour seul argument un autre objet du tableau.
La méthode de comparaison doit retourner :

- ✓ **NSOrderedAscending** si l'objet courant est plus petit que l'argument,
- ✓ **NSOrderedDescending** si l'objet courant est plus grand que l'argument,
- ✓ et **NSOrderedSame** si ils sont égaux.

Pour comparer des objets **NSString** sans prise en compte de la casse, la classe dispose de la méthode **localizedCaseInsensitiveCompare** qui retourne une énumération **NSComparaisonResult** initialisée avec une des 3 valeurs précédentes selon le résultat de la comparaison.

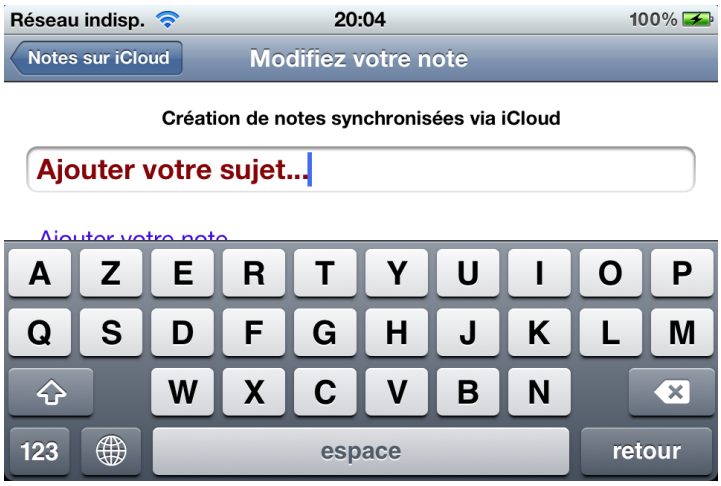
1. Ajouter à votre projet une classe nommée **NSURLwithComparator** héritant de **NSObject**.
2. Ajouter à cette classe une propriété nommée **noteURL** de type **NSURL***.
3. Ajouter à cette classe une méthode **- (id)initWithURL:(NSURL*)theURL** initialisant l'objet et la propriété **noteURL**.
4. Ajouter à cette classe une méthode de comparaison de prototype ci-dessous et coder celle-ci :
   -(NSComparisonResult)noteCompareURL:(NSURLwithComparator *)noteURLnext;

5. Modifier le projet de façon à remplacer l'usage de la classe **NSURL** par la classe **NSURLwithComparator** et ajouter le tri des URL dans la méthode **configAndStartQueryByBlock.**

6. Vérifier le fonctionnement du classement des notes par ordre croissant.

Partie 7 (extension possible) : La gestion du clavier sur iPhone.

Si vous disposez d'un iPhone (sinon sur la copie d'écran suivante) vous remarquerez que l'édition d'une note en mode paysage, n'est pas aisée pour la partie message compte tenu que le clavier masque la quasi totalité du contrôle **TextView**.



Le principe que je propose pour corriger ce problème consiste à capturer la notification associée à l'affichage du clavier, événement déclenché lorsque l'on sélectionne une zone d'édition (ici **TextField** ou **TextView**), puis au sein de la méthode de capture :

- ✓ lire l'information de durée configurée pour la transition du clavier, notée **TtransClavier**,
- ✓ lire la position origine en y du cadre du contrôle « Text Field » **noteSujet**, notée **offsetY**,
- ✓ déplacer la position origine en y du cadre de la vue de **–offsetY**, ceci avec une transition animée de durée **TtransClavier**.

Il faut faire en sorte que ce décalage n'ait lieu qu'à la condition que le matériel soit un iPhone et que l'interface soit en mode paysage.
Et voici ce que cela devrait donner :

1. Compléter la méthode **viewWillAppear** afin d'ajouter l'observation de la notification **UIKeyboardWillShowNotification**, avec pour méthode **keyboardWillShow:**, à la condition que le matériel soit de type iPhone. Le test du type de matériel peut être réalisé à l'aide de la classe **UIDevice** comme dans l'exemple ci-dessous :

```
if ([[[UIDevice currentDevice] model] hasPrefix:@"iPhone"]) {

}
```

2. Coder la méthode **keyboardWillShow:** de prototype ci-dessous selon le principe proposé :

```
- (void)keyboardWillShow:(NSNotification*)aNotification
```

L'orientation de l'interface peut être connue à l'aide de la méthode **interfaceOrientation** de la classe **UIViewController**.
Le test du mode paysage peut être réalisé à l'aide de la méthode **UIInterfaceOrientationIsLandscape**.
Exemple :

```
UIInterfaceOrientation orientation = [self interfaceOrientation];
    if (UIInterfaceOrientationIsLandscape(orientation))
    {

    }
```

Le paramètre **NSNotification** est un dictionnaire constitué d'un certain nombre de clés/valeurs associées aux types de notification, pour la liste des clés consulter l'aide sur UITextField.
Parmi ces clés, une indique le temps défini pour les animations du clavier, il s'agit de la clé **UIKeyboardAnimationDurationUserInfoKey**, pour laquelle on peut extraire la valeur à l'aide de la méthode **objectForKey**.

3. Vérifier le fonctionnement du glissement de la vue sur iPhone en mode paysage.

Annexe : Construction du projet de départ

Créer un projet du type « **Single View Application** » avec les paramètres :

- ✓ Product Name = **appliICloud**,
- ✓ Compagny Id = **net.marie-michel,**
- ✓ Utilisation du storyboard et d'ARC.

Ouvrir le Storyboard puis :

- ✓ Ajouter une nouvelle vue du type **Table View Controller**,
- ✓ Depuis le menu **Editor->Embeded in->Nav Controller** ajouter un « **Navigation Controller** » pour la vue précédente,
- ✓ Définir le « **Navigation Controller** » comme contrôleur de démarrage.

Ajouter une nouvelle classe Objective-C nommée **ViewControllerTableNotes** sans **xib** et de type « **subclass of UITableViewController** » et associer cette classe comme classe contrôleur de la vue « **Tab View Controller** ».

Dans le storyboard, vue « **Tab View Controller** » sélectionner le prototype de cellule « **Prototype cells** » puis :

- ✓ Définir le style à « **basic** »,
- ✓ Ajouter le fichier image **iconeAppXX.png** à votre projet et associer cette image à la cellule,
- ✓ Définir l'identificateur de cellule = « **noteCellID** ».

Définir les identificateurs des vues :

- ✓ Définir l'identificateur de la vue (storyboard ID) « **ViewController** » = **vueNoteID**,
- ✓ Définir l'identificateur de la vue « **ViewControllerTableNotes** » = **vueTableID**.

Ajouter un **UIBarButtonItem**, pour ceci :

- ✓ Sélectionner la vue « **Tab View Controller** » et lui ajouter un bouton **UIBarButtonItem** sur la droite de la barre de navigation,
- ✓ Modifier le texte du bouton avec « **Ajouter une note** », ou « **Add one note** »…,
- ✓ Ajouter une méthode **IBAction** nommée **ajoutNote** à la classe **ViewControllerTableNotes** pour la capture du clic sur le bouton et créer l'association.

Compléter la vue « **View Controller** », pour ceci :

- ✓ Ajouter à la vue les contrôles suivants positionnés de haut en bas :
  - Un contrôle **UILabel** avec le texte « Création de notes synchronisées via iCloud »,
  - Un contrôle **UITextField** qui servira à éditer le sujet de la note,
  - Un contrôle **UITextView** qui servira à éditer le message de la note.
- ✓ Ajouter les attributs et propriétés **noteSujet** et **noteMessage** à la classe « **ViewController** » et les associer aux contrôles précédents.

Ajouter à votre projet les fichiers icône d'application et images de chargement et les associer à votre projet.