TP Développement iOS 1 - Vos premières applications pas à pas.

IMT Lille Douai – UV AMSE

Lors de ces TP, vous avez accès à des Mac mini, avec XCode installé pour que vous puissiez développer puis envoyer vos applications soit sur vos iDevices, soit sur ceux de l'école. Vous vous identifierez sur le compte donné.
La première étape est donc de démarrer configurer votre certificat avant de démarrer XCode pour lequel l'icône est dans le dock.

1 Configuration du certificat

Vous avez reçu une invitation. Ainsi, vous pouvez créer un compte développeur Apple puis activer votre compte pour faire partie de l'équipe de développement. Une fois ceci fait, créez un certificat développeur, en suivant les instructions qui vous sont données dans votre interface de développeur.
Une fois ce certificat créé et approuvé, allez dans XCode, dans les paramètres, puis renseignez vos identifiant développeur Apple afin que vous puissiez être pris en compte comme développeur par XCode.

2 Votre première application

2.1 Création du projet

Pour ce premier projet, vous allez sélectionner un projet de type « Single View Application », « Universal » et donc qui s'exécutera de manière optimale autant sur iPhone que sur iPad. Ceci vous générera une application avec une fenêtre dans laquelle vous pourrez insérer les contrôles que vous voulez.

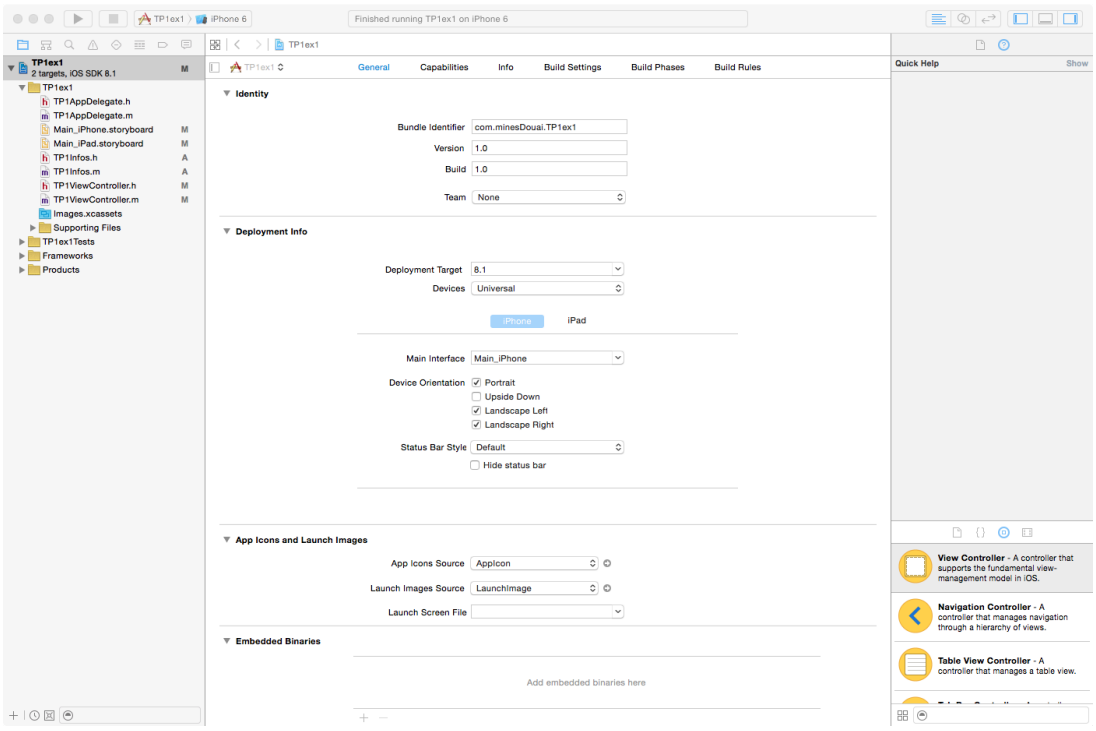

FIGURE 1 – Votre projet et sa configuration

Vous pouvez tout d'abord regarder la configuration de votre projet en cliquant, sur l'explorateur de gauche, sur le nom de celui-ci (la première ligne). Vous aurez ainsi les détails sur le nom de votre programme, son nom affiché sur l'iPhone, les orientations supportées, etc. (voir Fig. 1)
Votre projet au départ contient plusieurs fichiers, dont certains qui vous intéressent plus en détail :

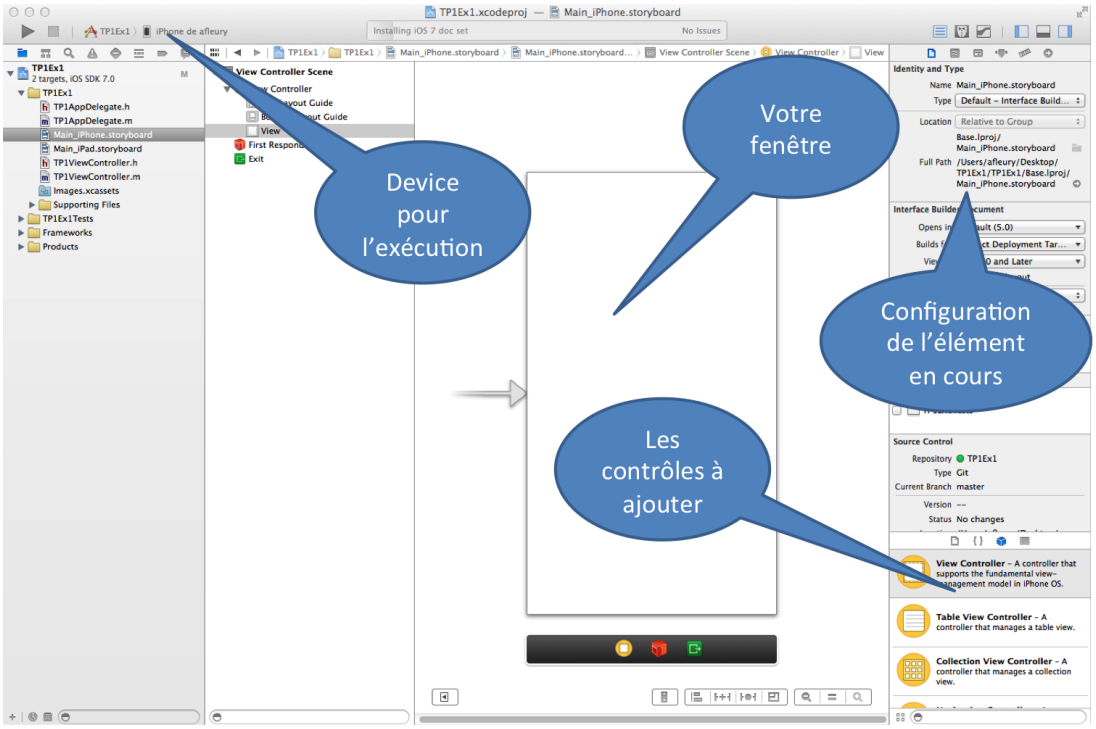

FIGURE 2 – Votre fenêtre de départ.

— Les fichiers MainStoryboard... qui contiennent les vues pour iPhone et pour iPad (différentes tailles d'écran) pour lequel la Fig. 2 vous montre la version iPhone,

— Les fichiers ViewController.h et ViewController.m qui décrivent la classe de la vue principale générée par XCode (que vous voyez dans le storyboard).

2.2 Votre première vue

Les fichiers storyboard sont indépendants pour iPhone et iPad. Vous devez donc construire les deux vues indépendamment l'une de l'autre. Par contre, la classe vue est la même (ViewController), ce qui permet d'écrire le comportement de la vue uniquement une fois (vous verrez le modèle MVC dans bien d'autres cours par la suite).

2.2.1 Remplissage de votre vue

La première étape est de remplir votre vue avec différents contrôles. Comme montré sur la figure 2, les contrôles se trouvent en bas à droite, vous y trouverez des labels (du texte), des zones de saisie (text edit), des contrôles multisegments, des boutons, etc.

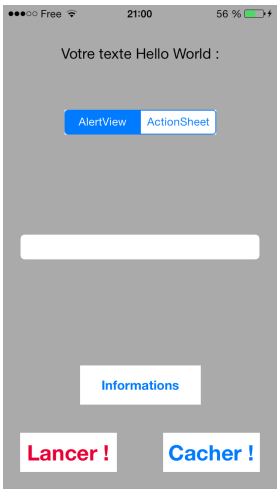

FIGURE 3 – Fenêtre de votre application

La figure 3 vous donne un exemple de présentation de votre vue, exécuté sur un iPhone, avec 3 boutons, un label, une zone de texte et un sélecteur multisegment.
Travail de cette partie : Ajoutez un à un les contrôles proposés à votre vue puis réglez les paramètres de la vue puis de chacun des contrôles pour donner à votre application le style que vous voulez. Faites de même pour la version iPad de la fenêtre.

2.2.2 Associer un élément de la vue à une variable ou une action

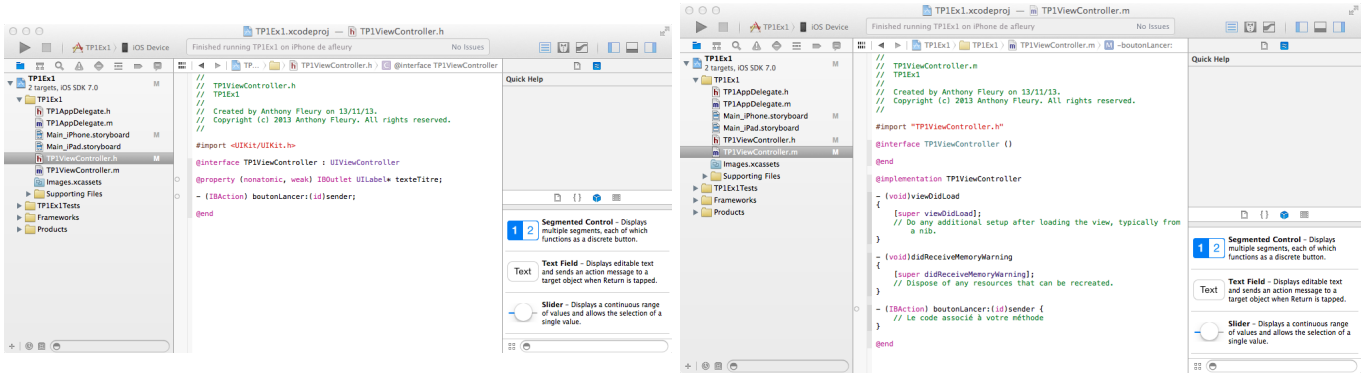

FIGURE 4 – Code de votre vue

L'association d'éléments de la vue à du code se fait en deux étapes. La première étape est d'insérer, dans le code de la vue, une nouvelle propriété qui décrit le contrôle. De la même manière, sont inclues aussi des méthodes d'instances qui vont réaliser des actions lors d'un évènement. La figure 4 montre la modification de la classe associée à la vue pour ajouter un label. IBOutlet est un mot particulier qui permet de désigner un élément d'interface graphique, IBAction est un type de retour désignant une méthode associée à un évènement graphique. Ces méthodes ont un type de retour (IBAction). Cependant, rien n'est retourné dans cette méthode (cas particulier). De plus, ces méthodes reçoivent en général l'envoyeur (déclaration de la forme −(IBAction)methode:(**id**)sender;) mais celui-ci peut-être ignoré lorsque ce n'est pas nécessaire, i.e. lorsque vous n'avez pas besoin de connaître le contrôle envoyant le message (vous en aurez besoin si une même méthode gère plusieurs évènements). À noter également un cas particulier par rapport à ce qui vous a été dit en cours. Logiquement, les propriétés associées à des pointeurs sont de type « strong ». Les IBOutlet font exception et ne sont que dans très peu de cas « strong ». Dans la plupart des cas ils seront « weak ».

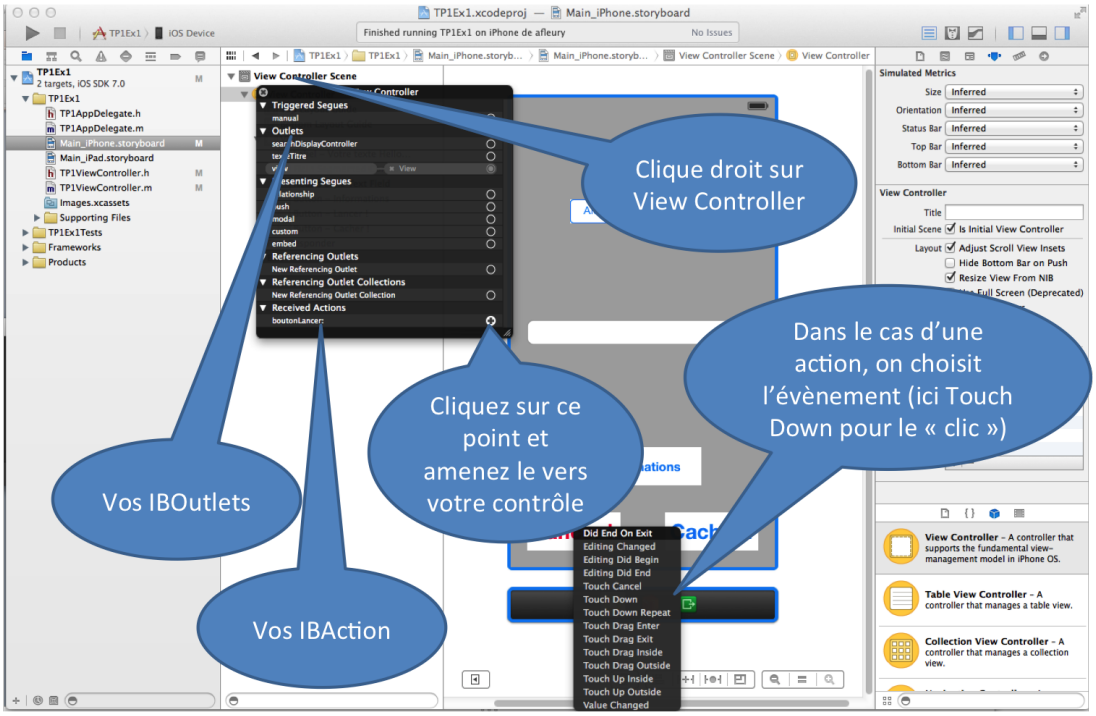

FIGURE 5 – Association variables ou méthodes avec un contrôle

La figure 5 vous montre l'association de vos contrôles avec les variables et les méthodes de votre classe gérant la vue. Un clic droit sur le contrôleur de vue du storyboard permet d'avoir accès aux Outlets et aux Actions.

Une fois ceci fait, dans le code de votre vue, vous pouvez alors avoir accès aux éléments de vos contrôles et aussi modifier la manière dont ils sont affichés. Vos contrôles ont en effet des propriétés membres, avec les accesseurs qui sont créés, et qui permettent d'avoir des informations sur celui-ci ou modifier ces informations. Ainsi, tous les contrôles auront une propriété hidden, et un setter setHidden :(BOOL) (en envoyant YES ou NO), permettant de cacher, montrer ou de savoir si le contrôle est caché ou non. De manière plus particulière, un UITextField* (zone de saisie de texte) aura une propriété text et un setter setText :(NSString*) permettant d'avoir accès au texte tapé et de modifier le texte de la zone. Un UIButton* aura lui un setTitle alors qu'un UISegmentedControl aura une propriété selectedSegmentIndex renvoyant un entier et un setter setSelectedSegmentIndex pour modifier celui qui est sélectionné. Par exemple, pour un contrôle UITextField* champTexte, écrire [champTexte setText:@"Nouveau␣Texte"]; changera le texte affiché par le contrôle.

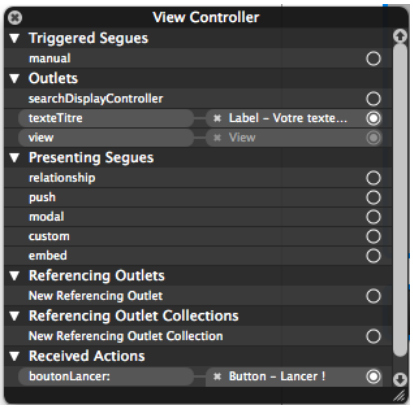

FIGURE 6 – La fenêtre une fois les associations faites

Une fois ceci fait, vous voyez, comme sur la Figure 6, les différentes associations entre les variables et vos contrôles.
Travail de cette partie : Ajoutez les différentes variables pour vos contrôles, et une méthode d'instance que vous associerez au bouton « Cacher ». Cette méthode cachera l'ensemble des éléments de la fenêtre graphique (sauf le bouton cacher et le bouton informations) puis changera le texte pour afficher Montrer. Un autre clic fera l'inverse (montrer tous les contrôles et rechanger le texte du bouton pour Cacher).
Associez de la même manière les éléments de votre fenêtre iPad au mêmes variables et actions. Vous n'écrivez ainsi qu'une fois les actions de votre vue pour les deux types de fenêtres.

2.2.3 Votre bouton lancer

Nous voulons que ce bouton affiche une boîte de dialogue qui répète le message inscrit. Pour ceci, nous allons utiliser deux nouveaux types UIAlertView et UIActionSheet. Il est à noter que dans ce sujet, UIAlertView et UIActionSheet ont été conservés pour leur simplicité. iOS 8 (et XCode 6) introduit une interface unifiée, UIAlertController, un peu plus compliquée à utiliser de prime abord.
Pour utiliser ceux-ci, vous allez devoir répondre, dans votre vue, à certains messages de ces deux types. Ceci sera implémenté ainsi :

Listing 1 – Fichier .h de votre classe Vue

```
// ...
@interface ViewController : UIViewController <UIAlertViewDelegate,
        UIActionSheetDelegate>
{
        //...
        UIAlertView* message;
        UIActionSheet* action;
}
- (void) chargerMessage;
- (void) chargerAction;

- (void) alertView:(UIAlertView*) alertView clickedButtonAtIndex:(NSInteger)
        buttonIndex;
- (void) actionSheet:(UIActionSheet*) actionSheet didDismissWithButtonIndex:(
        NSInteger)buttonIndex;
```

Listing 2 – Fichier .m de votre classe Vue

```
// ...
@implementation ViewController

// ...

- (void) chargerMessage
{
        message = [[UIAlertView alloc] initWithTitle:@"Titre"
                message:@"Votre Message"
                delegate:self
                cancelButtonTitle:@"Titre du bouton"
                otherButtonTitle:@"titre 1", @"titre 2", ..., nil];
        [message show];
}

- (void) chargerAction
{
        action = [UIActionSheet alloc] initWithTitle:@"Titre"
                delegate:self
                cancelButtonTitle:@"Titre" // ou nil
                destructiveButtonTitle:@"Titre Bouton Rouge"
                otherButtonTitle:@"titre 1", @"Titre 2", ..., nil];
        [action showInView:self.view];

}

- (void) alertView:(UIAlertView*) alertView clickedButtonAtIndex:(NSInteger)
      buttonIndex
{
        // la fenêtre a quitté (celle-ci est désignée par la variable alertView pour
              savoir quelle fenêtre est concernée)
        // le bouton ayant servi à quitter est buttonIndex (index de 0 au nombre de
              boutons - 1
        // faites ici les actions à entreprendre dans les différents cas.
}

- (void) actionSheet:(UIActionSheet*) actionSheet didDismissWithButtonIndex:(
      NSInteger)buttonIndex
{
        // la fenêtre a quitté (celle-ci est désignée par la variable actionSheet
              pour savoir quelle fenêtre est concernée)
        // le bouton ayant servi à quitter est buttonIndex (index de 0 au nombre de
              boutons - 1
        // faites ici les actions à entreprendre dans les différents cas.
}
```

Ce code déclare des variables des bons types (pour des fenêtres d'alertes et des fenêtres d'actions). Elle déclare aussi le fait que la vue va se conformer aux protocoles de UIAlertViewDelegate et UIActionSheetDelegate, c'est à dire que la vue va répondre aux messages envoyés par ces fenêtres.
Pour ceci, on implémente une méthode par types (méthodes qui sont demandées par le protocole à suivre), qui sont les méthodes lorsque les fenêtres se ferment.
Un exemple pour construire et appeler chacune d'entre-elle vous est aussi montré dans ce code. Selon ce que vous voulez faire avec vos fenêtres, vous mettrez du code différent dans celles-ci.
Travail de cette partie : Adaptez le code donné pour que lorsque vous allez cliquer sur le bouton lancer, selon si AlertView ou ActionView est sélectionné, vous chargez le bon type de fenêtre, avec comme message le texte que vous avez entré et deux boutons : un bouton qui va uniquement fermer la fenêtre (ne rien faire), un autre bouton qui va supprimer le texte de la zone d'édition.

2.2.4 Comportement spécifique : l'iPhone et le clavier...

Vous avez donc maintenant écrit le code de votre bouton Lancer et celui de votre bouton Cacher/Montrer. Lorsque vous tapez du texte, le clavier apparait automatiquement dans la fenêtre. Dans le cas où vous êtes sur iPad, pas de soucis, un bouton permettant de cacher le clavier sur iPad est disponible. Par contre, sur iPhone ce n'est pas le cas. Nolus voudrions faire disparaitre le clavier en appuyant sur retour.
Pour faire ceci, nous allons définir une méthode −(IBAction) finSaisie :(**id**)sender contenant comme seule instruction : [sender resignFirstResponder];.
Travail de cette partie : Insérez ce code dans votre vue puis associez cette action à l'évènement « Did End On Exit » de votre zone de texte. Vérifiez alors que le clavier disparait après la saisie.

2.2.5 Des tests supplémentaires ?

Soit sur le simulateur soit sur un iDevice, testez votre fenêtre en portrait puis en paysage. Que constatez-vous ? En cas de soucis, à ce moment là, contactez l'enseignant pour plus d'explications.

2.3 Ajout d'une vue

Maintenant que vous avez écrit votre première vue et bien décrit son comportement, nous allons voir comment construire et appeler une seconde vue.

2.3.1 Ajout d'une vue au storyboard

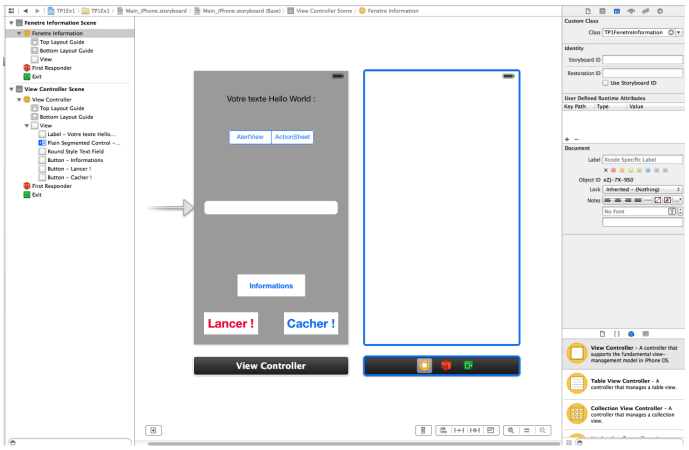

FIGURE 7 – Création et Association de votre vue

Dans votre storyboard, vous pouvez ajouter un objet de type View Controller (voir Figure 7). Cet objet va gérer une nouvelle fenêtre dans laquelle vous pouvez ajouter des contrôles.
Travail de cette partie : Faites ceci pour votre storyboard iPhone puis iPad.

2.3.2 Création de la classe associée

Dans Files, New, File, sélectionnez un type Cocoa Touch pour iOS (sur la gauche) puis sur la droite un type Objective C class. Le subclass est à mettre à UIViewController, et vous devez donner un nom à votre classe dans Class. Laissez désélectionnées les deux autres options. Sur la fenêtre suivante, cliquez sur Create.
Travail de cette partie : Regardez de nouveau la figure 7 pour associer cette classe nouvellement créée à votre vue dans le storyboard.

2.3.3 Association de l'ouverture de la vue à un bouton

Suivez la figure 8 pour associer l'ouverture de votre nouvelle vue au clic du bouton information. Lorsque cela vous est demandé, vous pouvez choisir un mode d'ouverture « modal ».
Travail de cette partie : Faites cette association avec votre bouton information pour votre storyboard iPhone puis iPad.

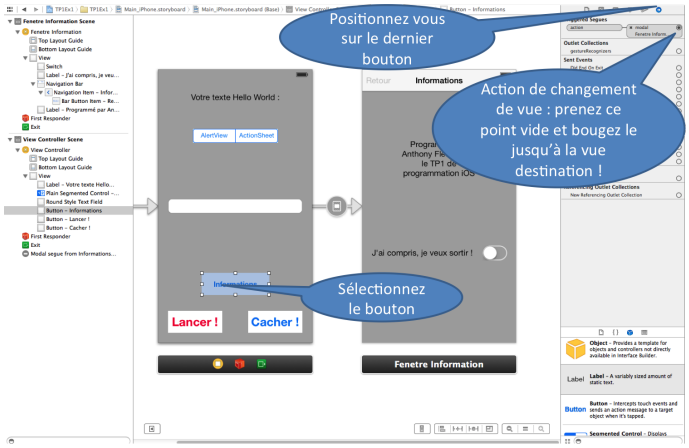

FIGURE 8 – Changement de vue au clic sur le bouton !

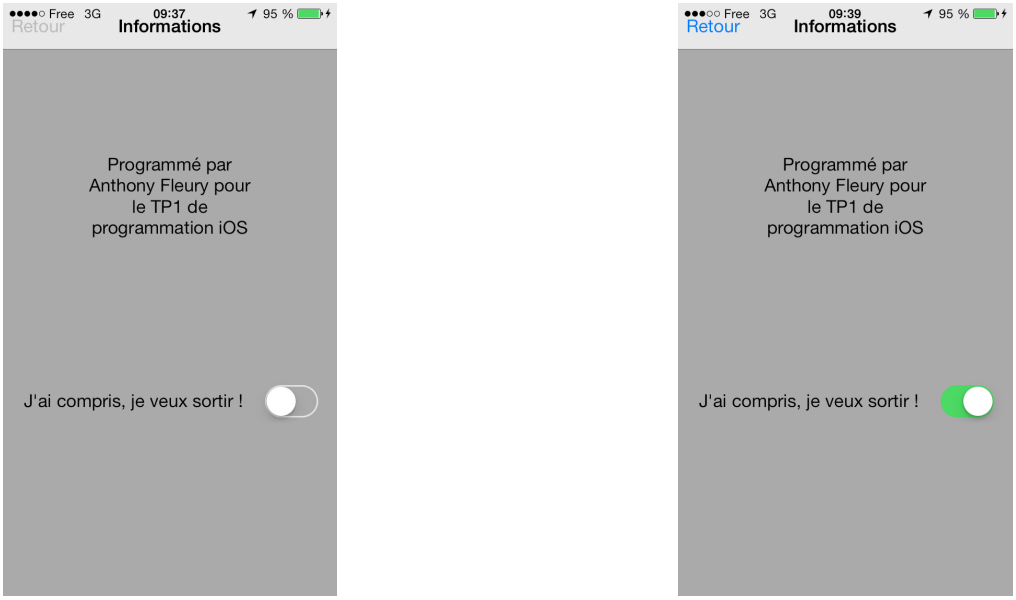

FIGURE 9 – La seconde vue.

2.3.4 Comportement de cette vue

Cette seconde vue vous est montrée sur la Figure 9. Dans cette vue nous avons :

- Une barre de navigation avec le titre informations,
- Un texte donnant les informations sur l'application,
- Un texte et un switch qui va permettre de valider le fait que le texte précédent a été compris,
- Un bouton retour dans la barre de navigation.

Ce que l'on veut est que lorsque le switch est sur on (faites une méthode qui se déclenche sur un évènement changement de valeur du bouton de barre de navigation), alors le bouton est actif. Par défaut dans la vue, le bouton est inactif (enabled décoché) et le switch sur off.
L'appuie sur le bouton retour va lancer le code qui permet de fermer la fenêtre en cours et qui est :
`[self dismissViewControllerAnimated:YES completion:nil];`.
Attention, pour accéder à la valeur du switch, si celui-ci s'appelle monSwitch, alors vous devez utiliser monSwitch.on.
Pour le bouton de barre de navigation, bouton.enabled=YES; l'active alors que bouton.enabled=NO; le désactive.
Travail de cette partie : Vérifiez le comportement global de toute votre application et si vous voulez, vous pouvez également changer les différents paramètres de celle-ci (icône, splash window, etc.).

3 Votre seconde application : un petit jeu

Laissez maintenant parler votre imagination et écrivez un petit jeu, pour lequel vous allez définir quelques questions dans des tableaux de type NSArray par exemple, avec des bonnes et des mauvaises réponses, puis une validation pour

chaque question (qui sera tirée au hasard).
Faites un jeu avec un système de score (prenez exemple sur des jeux TV ou autre).
À vous de jouer et de coder ! N'hésitez pas à poser des questions pour réaliser certaines fonctionnalités que vous ne sauriez pas écrire.

4 Références

Le site d'Apple contient de très nombreuses documentations (en ligne et sous forme de PDF) pour l'ensemble du SDK iOS. Vous les trouverez à l'adresse `http://developer.apple.com/library/ios/index.html`.