6

Les contrôles spécifiques

Au sommaire de ce chapitre

- Interagir avec des contrôles spécifiques : la création d'un utilitaire
- Interagir avec des réglettes, des alertes, des contrôles segmentés et des sous-vues
- Intégrer une roulette à l'application
- Pour aller plus loin

Si certaines applications très populaires tiennent en une poignée d'éléments intelligemment agencés dans une fenêtre unique, la plupart des projets les plus ambitieux reposent sur une multitude de vues qui s'adaptent aux attentes des utilisateurs. Il n'est pas rare d'imbriquer quatre ou cinq fenêtres différentes, elles-mêmes constellées de boutons, libellés ou champs de texte. Pour préserver un contrôle intuitif malgré la multiplication des objets d'interface, le SDK de l'iPhone dispose d'une bibliothèque très riche : réglettes, interrupteurs, roulettes,

etc. Tous ces éléments autorisent un contrôle minutieux et permettent aux utilisateurs d'interagir avec votre application de manière ergonomique. À l'image du motif de conception MVC, votre tâche ne consiste pas seulement à vous focaliser sur la logique interne de vos programmes ; vous devez également agencer au mieux tous ces éléments, sans pour autant céder à la facilité de déposer pêle-mêle tout type d'objets. Votre application doit toujours pleinement justifier l'utilisation de tous ces contrôles avancés.

Nous l'avons vu au chapitre précédent : si l'écran de l'iPhone présente nécessairement un affichage limité, les systèmes de navigation (onglets, boutons, barres d'outils, etc.) vous aident à regrouper correctement vos éléments et à ne pas alourdir l'interface. L'immense famille des utilitaires gagne à s'articuler autour de tels systèmes, afin de préserver la lisibilité de l'application. Au cours de ce chapitre, nous verrons comment aboutir à de tels projets autour de ces contrôles avancés ; ces derniers vous font gagner un temps précieux et élargissent considérablement le champ de vos possibilités tout en présentant une interface conviviale aux utilisateurs.

Face à de tels objectifs, les roulettes jouent un rôle prépondérant. Elles constituent l'une des interfaces-clés de l'iPhone : on navigue verticalement parmi leurs éléments, en les faisant glisser au doigt. Numéros de téléphone, paramètres d'une application ou contenu extrait d'un service web : il existe de nombreuses manières d'agencer les éléments qu'elles contiennent et de les rendre interactives. Elles deviennent alors la colonne vertébrale de toute votre application et donnent accès à une immense variété de services et d'informations. Au cours de ce chapitre, nous étudierons leur intérêt et nous apprendrons à les disposer sur l'écran de l'iPhone à travers une série d'exemples concrets.

Interagir avec des contrôles spécifiques : la création d'un utilitaire

Le modèle Utility Application

Tel que le conçoit Xcode, les utilitaires sont des applications mono-tâches qui servent un but précis : ce sont des outils simples que les utilisateurs déclenchent pour répondre à un besoin spécifique (voir Figure 6.1). Bulletin météo, calcul d'itinéraires, outil de bricolage, etc., tous ces projets s'articulent autour d'une vue principale, qui offre un accès direct à l'information, et d'une vue secondaire accessible à l'aide d'un petit bouton "i". Vous l'avez probablement déjà repéré sur vos applications préférées, à commencer par le programme Bourse qui est préinstallé sur l'iPhone : ce bouton déclenche une animation de transition qui fait pivoter la vue principale vers cette vue secondaire, à travers laquelle on configure sommairement

l'utilitaire ou l'on accède à des informations complémentaires. Vous pouvez rapidement développer une application de ce type grâce à l'un des modèles intégrés à Xcode.

Figure 6.1

L'application GuitarToolkit *vous permet d'accorder à une guitare et d'accéder à une page d'information consacrée au développeur.*

Nous allons créer une application de ce type en ajoutant une série de contrôles avancés à la vue principale (interrupteurs, boutons, champs de texte, etc.) et en décrivant le projet dans la vue secondaire. Sous Xcode, déroulez le menu File > New Project, puis choisissez le modèle "Utility Application". Cliquez ensuite sur le bouton Choose et saisissez "Motdepasse" en guise de nom de projet. Notre utilitaire est un générateur de mot de passe : en fonction de critères sélectionnés par l'utilisateur (emploi de majuscules, minuscules, chiffres et/ou signes de ponctuation, longueur du mot de passe, etc.), l'application propose aléatoirement une chaîne de caractères qu'on pourra employer lors de la création d'un compte auprès d'un service web, par exemple. Contrairement à notre exemple de quiz, où l'on devait presser directement le seul bouton correspondant à notre réponse, il s'agit ici de réunir tous les critères définis par l'utilisateur et de les traiter d'un seul tenant à l'aide d'une action.

Après avoir créé votre nouveau projet, essayez de le compiler en pressant simultanément les touches Cmd+R. Le Simulateur d'iPhone présente cette première ébauche : votre utilitaire s'articule autour de deux vues vides que l'on permute en pressant un petit bouton "i" (voir Figure 6.2). Depuis la seconde vue, on revient vers la vue principale en cliquant sur un bouton figurant dans la barre de titre – Xcode a donc mis en place une application autour d'un système de navigation sommaire, qui est une instance de la classe `UINavigationBar`.

Figure 6.2

Le comportement initial d'un utilitaire créé sous Xcode.

Auscultez de plus près les fichiers créés automatiquement. Leur agencement obéit au principe suivant :

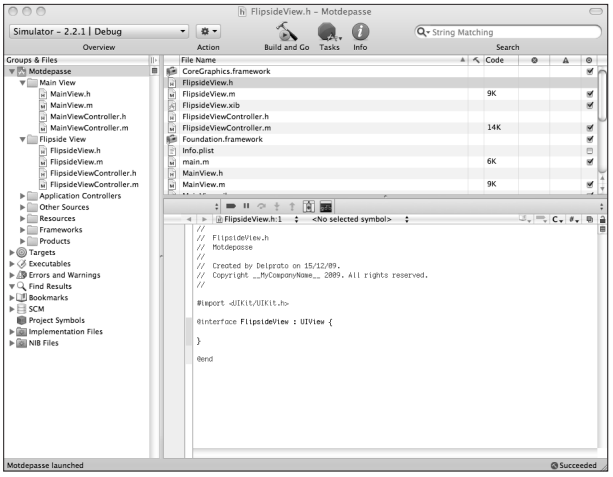
- Comme l'indique la liste des propriétés `Info.plist`, le fichier nib par défaut est `MainWindow`. Il est ainsi chargé en premier.
- En observant ce fichier, on constate qu'il intègre une instance de la classe `RootViewController`. Il s'agit d'une sous-classe d'`UIView` qui ne dispose que d'un outlet, `infoButton`, et d'une action, `toggleView`. Le premier est un bouton affiché par défaut sur l'interface qui déclenche l'action : la vue principale permute vers la vue secondaire, selon le principe général que nous avons exposé au chapitre précédent.
- La classe `RootViewController` gère donc le changement de vue. On lui associe par défaut deux contrôleurs supplémentaires : `MainViewController` et `FlipSideViewController`, qui correspondent respectivement à la vue principale et secondaire. En parallèle, cette classe met en place une instance de la classe `UINavigationBar` que l'on utilise à travers l'action `toggleView` pour gérer le changement de vue. Comme vous le constatez en ouvrant le fichier `RootViewController.m`, c'est la vue principale qui est chargée par défaut : `initWithNibName:@"MainView"` indique que la vue `MainView.xib` s'affiche dès l'ouverture de l'application.
- Cette vue est pour l'instant vide. Elle ne comprend qu'une instance de la classe `MainView`, elle-même sous-classe d'`UIView`. Son contrôleur `MainViewController` est lui aussi vide et s'apprête à recueillir tous les traitements que vous souhaitez faire figurer sur la vue principale de votre utilitaire.

- L'action `toggleView` de la classe `RootViewController` gère le chargement de la vue secondaire. Elle commence par vérifier la vue qui figure actuellement sur l'écran de l'iPhone. S'il ne s'agit pas de la vue secondaire, on appelle la méthode `loadFlipsideViewController` et on charge le second fichier nib, `FlipSideView.xib`. Cette méthode exploite ensuite la barre de navigation instanciée dans la fenêtre principale de l'application, `MainWindow.xib`, pour ajouter un titre à la seconde vue et permettre le retour à la vue principale.
- L'interface de cette seconde vue est donc définie par le fichier `FlipsideView.xib`. Là encore, la fenêtre est vide et ne comprend qu'une instance de la classe `FlipSideView`. Son contrôleur `FlipsideViewController` est également vide et s'apprête à héberger toutes les actions déclenchées sur la seconde vue de votre utilitaire.

Si tout ceci vous paraît encore abstrait, parcourez en détail le code source et retenez ce concept essentiel : pour personnaliser rapidement votre utilitaire, vous devez retoucher les deux vues successives créées par Xcode (`MainView` et `FlipsideView`), adapter leur classe contrôleur et éventuellement personnaliser la barre de titre qui soutient leur basculement. Vous avez également la possibilité d'ajouter des traitements personnalisés au lancement de votre utilitaire grâce au délégué de l'application, `MotdepasseAppDelegate`. Même s'il apparaît en anglais, le nom des classes est suffisamment explicite pour mettre rapidement le pied à l'étrier : vous profitez bel et bien du canevas d'une application multivues en quelques clics de souris (voir Figure 6.3).

Figure 6.3

L'agencement initial des classes créées par Xcode dans notre projet d'utilitaire.



Le cahier des charges et l'ébauche préliminaire

Plaçons-nous dans le contexte d'un projet réel. Avant de mettre les mains dans le cambouis et d'ajouter nos éléments d'interface et nos actions, nous devons rédiger un bref cahier des charges et définir les interactions sur papier. Les mots de passe trop évidents sont aujourd'hui devenus la cible préférée des pirates, qui en profitent pour accéder à la quasi-totalité de nos activités. Il n'est en effet pas rare de décliner le sempiternel même sésame à l'ensemble de nos services web (courrier électronique, réseaux sociaux, services en ligne, etc.), ce qui les expose dangereusement. Notre utilitaire vise à générer aléatoirement des mots de passe beaucoup plus complexes, qui mêlent éventuellement des majuscules et des minuscules mais aussi des chiffres et des signes de ponctuation. Lorsqu'un utilisateur s'apprête à créer un compte sur un forum ou un service en ligne, il lui suffira de dégainer son iPhone et d'utiliser notre application pour s'assurer de choisir un mot de passe inviolable.

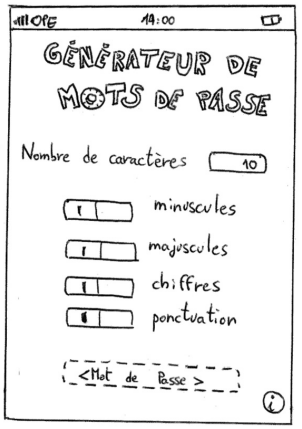
En dégageant ainsi le rôle essentiel de notre application, nous avons identifié les principaux contrôles que nous devrons disposer sur l'interface :

- un champ de saisie pour indiquer le nombre de caractères composant le mot de passe ;
- des contrôles permettant de complexifier le mot de passe et d'y faire figurer des caractères majuscules et minuscules, des chiffres et des signes de ponctuation ;
- un bouton pour déclencher la création du mot de passe ;
- un champ affichant le résultat ;
- un bouton "i" pour basculer vers la vue secondaire ;
- une seconde vue où l'on met en garde l'utilisateur contre le risque d'employer des mots de passe trop évidents et à travers laquelle on réalise la "publicité" de notre gamme de produits.

Dans ce type de projet, le dialogue homme-machine est réduit à sa plus simple expression : l'utilisateur personnalise une série de valeurs et clique sur un bouton unique afin de lancer le traitement. En retour, notre application affiche le résultat escompté. Des pressions supplémentaires sur le bouton génèrent de nouveaux mots de passe : l'utilisateur peut ainsi rapidement parcourir une série de propositions et choisir celle qui lui paraît la plus efficace.

Parmi les éléments à notre disposition, il nous apparaît clairement que les interrupteurs (classe `UISwitch`) constituent la solution idéale pour choisir les éléments devant figurer dans les mots de passe. On active ou non l'ajout de signes de ponctuation, par exemple. Il s'agit d'un contrôle emblématique de l'iPhone. Les utilisateurs s'y sont habitués dès l'écran des réglages principaux et ils comprendront d'emblée son rôle et son fonctionnement. Sur papier, nous essayons d'agencer au mieux tous ces éléments qui figureront dans la vue principale de l'application. Le premier résultat apparaît à la Figure 6.4.

Figure 6.4
Une première ébauche de l'interface.



Cette première esquisse révèle quelques défauts. En figurant par défaut sous les contrôles, le champ contenant le mot de passe généré risque d'alourdir l'affichage. Il s'agit de l'information principale renvoyée par notre application : elle doit figurer au premier plan. Afin d'éviter que l'utilisateur tente d'y saisir quoi que ce soit, nous réglerons sa couche alpha à 0 (le champ de texte deviendra ainsi totalement transparent) et nous le ferons apparaître progressivement au moment de générer un mot de passe. Les différents interrupteurs doivent être remisés au second plan ; si les utilisateurs vont essayer de combiner des critères distincts lors de leur première découverte de l'application, il y a fort à parier qu'ils vont maintenir la même nomenclature par la suite. Nous aboutissons ainsi à une seconde esquisse sur papier, plus proche du résultat escompté (voir Figure 6.5).

Figure 6.5
On modifie l'agencement des contrôles, pour gagner en clarté et en lisibilité.

Dans le cadre d'applications complexes, il n'est pas futile de réfléchir sur papier aux algorithmes à mettre en place. Notre générateur de mots de passe ne compte qu'une action :

celle-ci récupère la valeur des interrupteurs, crée une longue chaîne de caractères épousant les souhaits de l'utilisateur, puis sélectionne aléatoirement une série de caractères issus de cette chaîne. Le résultat final est ensuite présenté à l'utilisateur, qui le découvre à travers un champ de texte apparaissant progressivement. Nous avons ainsi besoin de quatre interrupteurs, correspondant aux différents critères, de deux champs de texte pour saisir la longueur du mot de passe et afficher le résultat, et d'un bouton pour lancer l'opération.

La logique interne des contrôleurs

Comme nous l'avons vu, il n'est pas nécessaire de retoucher la classe `RootViewController` pour aboutir à un utilitaire fonctionnel : Xcode a préparé tous les fichiers nécessaires et a tissé les liaisons entre les différents éléments de base. Pour personnaliser la barre de titre figurant au-dessus de la vue secondaire, nous devons toutefois retoucher un aspect de l'implémentation de cette classe. Modifiez le fichier `RootViewController.m` en ajoutant les deux lignes apparaissant en gras au Listing 6.1.

Listing 6.1 : *RootViewController.m*

```
#import "RootViewController.h"
#import "MainViewController.h"
#import "FlipsideViewController.h"

@implementation RootViewController

@synthesize infoButton;
@synthesize flipsideNavigationBar;
@synthesize mainViewController;
@synthesize flipsideViewController;

- (void)viewDidLoad {

    [super viewDidLoad];
    MainViewController *viewController = [[MainViewController alloc]
                        initWithNibName:@"MainView" bundle:nil];
    self.mainViewController = viewController;
    [viewController release];

    [self.view insertSubview:mainViewController.view
          belowSubview:infoButton];
}

- (void)loadFlipsideViewController {

    FlipsideViewController *viewController = [[FlipsideViewController
          alloc] initWithNibName:@"FlipsideView" bundle:nil];
    self.flipsideViewController = viewController;
    [viewController release];
```

```
    // Set up the navigation bar
    UINavigationBar *aNavigationBar = [[UINavigationBar alloc]
          initWithFrame:CGRectMake(0.0, 0.0, 320.0, 44.0)];
    aNavigationBar.barStyle = UIBarStyleBlackOpaque;
    self.flipsideNavigationBar = aNavigationBar;
    [aNavigationBar release];

    UIBarButtonItem *buttonItem = [[UIBarButtonItem alloc]
          initWithTitle:@"retour" style:UIBarButtonItemStylePlain
          target:self action:@selector(toggleView)];
    UINavigationItem *navigationItem = [[UINavigationItem alloc]
          initWithTitle:@"À propos"];
    navigationItem.rightBarButtonItem = buttonItem;
    [flipsideNavigationBar pushNavigationItem:navigationItem
          animated:NO];
    [navigationItem release];
    [buttonItem release];
}

- (IBAction)toggleView {
    /*
     This method is called when the info or Done button is pressed.
     It flips the displayed view from the main view to the flipside view
     and vice-versa.
     */
    if (flipsideViewController == nil) {
        [self loadFlipsideViewController];
    }

    UIView *mainView = mainViewController.view;
    UIView *flipsideView = flipsideViewController.view;

    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:1];
    [UIView setAnimationTransition:([mainView superview] ?
          UIViewAnimationTransitionFlipFromRight :
          UIViewAnimationTransitionFlipFromLeft) forView:self.
          view cache:YES];

    if ([mainView superview] != nil) {
        [flipsideViewController viewWillAppear:YES];
        [mainViewController viewWillDisappear:YES];
        [mainView removeFromSuperview];
        [infoButton removeFromSuperview];
        [self.view addSubview:flipsideView];
        [self.view insertSubview:flipsideNavigationBar
                aboveSubview:flipsideView];
        [mainViewController viewDidDisappear:YES];
        [flipsideViewController viewDidAppear:YES];

    } else {
```

```
        [mainViewController viewWillAppear:YES];
        [flipsideViewController viewWillDisappear:YES];
        [flipsideView removeFromSuperview];
        [flipsideNavigationBar removeFromSuperview];
        [self.view addSubview:mainView];
        [self.view insertSubview:infoButton
                aboveSubview:mainViewController.view];
        [flipsideViewController viewDidDisappear:YES];
        [mainViewController viewDidAppear:YES];
    }
    [UIView commitAnimations];
}

/*
// Override to allow orientations other than the default portrait
// orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
        interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Releases the view if it doesn't have a superview
    // Release anything that's not essential, such as cached data
}

- (void)dealloc {
    [infoButton release];
    [flipsideNavigationBar release];
    [mainViewController release];
    [flipsideViewController release];
    [super dealloc];
}

@end
```

Comme vous pouvez le constater, le contrôleur racine commence par charger la vue "MainView" (soit `MainView.xib`) dans l'action `viewDidLoad` qui s'exécute automatiquement au lancement de l'application. L'action `loadFlipsideViewController` charge la seconde vue lorsque l'on presse le bouton "i" figurant en bas à droite de l'écran. Nous sommes essentiellement intervenus lors de l'initialisation de la barre de navigation, en modifiant le titre de la seconde vue et le libellé figurant sur le bouton de retour :

```
UIBarButtonItem *buttonItem = [[UIBarButtonItem alloc]
   initWithTitle:@"retour" style:UIBarButtonItemStylePlain target:self
   action:@selector(toggleView)];
```