Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de bachelor 2010

Filière Informatique de gestion

IPos

Etudiant : Cheseaux Francois

Professeur : Glassey Nicole

Résumé

Ce travail de diplôme s'inscrit dans le cadre du projet MMHS (My Mobile Heath Space) qui a pour objectif de mettre à disposition du monde de la santé des technologies mobiles et des réseaux sociaux pour favoriser l'autogestion par le patient.

Dans cet optique, une recherche a été effectuée afin de fournir plusieurs moyens d'apprentissage en utilisant les possibilités techniques d'un Iphone, le SmartPhone d'Apple.

La maladie de l'asthme a été prise comme modèle de référence. Les réalisations s'inspirent librement de documents traitant sur le sujet et n'ont pas encore fais l'objet d'un validation par des professionnels.

Le choix s'est porté sur deux applications principales :

IPos Simulator

Afin d'utiliser au maximum les possibilités d'interactions offertes par l'IPhone et toujours sur le modèle de l'asthme, un simulateur de ventolin a été développé.

Se basant sur le mode d'emploi de l'appareil, il permet d'effectuer toutes les opérations nécessaires à sa bonne utilisation.

IPos Quizz

Dans l'optique d'apprendre à connaître une maladie, un modèle de jeu de quizz a été développé. Il permet de mettre à disposition plusieurs séries de questions par niveau ou par thème.

Une partie optionnelle a également pu être développée :

IPos Action

Dans le cadre d'un traitement pour une maladie chronique, des plans d'action sont définis par le médecin du patient. Ceux-ci contiennent des zones d'état auquel le patient peut s'identifier par le biais de symptômes et accomplir des actions afin d'améliorer sa situation.

Une interface de consultation simple et une zone de configuration ont ainsi été créées.

Table des matières

Introduction

Ce travail de diplôme s'inscrit dans le cadre du projet MMHS (My Mobile Heath Space), E-Posologie2. L'objectif est d'amener un service mobile innovant dans le cadre de la prévention de la santé et de la prise en charge personnelle du patient.

Ainsi, le projet EP2 vise à mettre à disposition du monde de la santé les apports des technologies mobiles et des réseaux sociaux pour favoriser l'autogestion par le patient. Le public visé dans le cadre de ce projet regroupe les jeunes de 8 à 18 ans, organisés en deux tranches : les enfants (8-12 ans) et les adolescents/jeunes adultes (13-18 ans).

Le comportement, les besoins et le rapport du patient à sa santé évoluent. Internet, le téléphone mobile, les réseaux sociaux font désormais partie intégrante du processus thérapeutique et ce projet vise à les intégrer en offrant aux patients :

- **L'éducation thérapeutique**, qui permet de rendre le patient acteur de ses soins, il s'agit de l'aider dans sa formation pour prendre correctement ses médicaments.

- **Le suivi et l'accompagnement thérapeutique**, (spécialement dans un contexte de maladie chronique), qui permet de rassurer, de vérifier les données, d'être présent et de donner des conseils.

Le but de ce travail de diplôme est de proposer un prototype montrant les possibilités de l'Iphone dans le contexte décrit.

Dans cette première démarche, aucune validation n'a été effectuée par des professionnels. Il s'agit d'une inspiration libre de diverses informations médicales.

Présentation générale

Aide à la lecture

Pour une meilleure compréhension de la technique générale du développement d'application sur Iphone, un tutoriel, effectué par M. Guillaume Fort, est disponible sur le cd joint au dossier.

De plus, le cahier des charges complet est mis en annexe du présent dossier.

Objectifs généraux

L'objectif principal de ce travail de diplôme est d'utiliser les possibilités offertes par l'IPhone afin de permettre au malade d'apprendre à connaître sa maladie et à gérer sa posologie de manière ludique.

La suite d'application IPos s'adresse avant tout à des enfants âgés de 8 à 12 ans. Le jeu a été choisi comme canal de communication afin de permettre un apprentissage attractif de la maladie et de la manière de la gérer. Un soin tout particulier a été apporté aux interfaces et au contenu afin de coller au mieux au public cible.

L'exemple de l'école de l'asthme de Sion a été pris comme modèle. Durant 2 périodes de 1h30, les enfants asthmatiques y apprennent à gérer leur maladie en suivant des cours donnés par un professionnel.

Les applications se basent sur ce modèle mais peuvent être déclinées afin de correspondre à n'importe quelle maladie demandant une posologie régulière.

Aucune validation n'a été effectuée auprès de professionnels, il s'agit là d'une adaptation libre, servant uniquement à démontrer les possibilités de l'Iphone dans ce contexte.

Objectifs technologiques

La plateforme choisie pour ce travail est l'IPhone, le Smartphone d'Apple. Le projet se composera de deux applications natives.

Aucune connexion extérieure n'est prévue, les applications seront donc complètement indépendantes. Elles devront cependant permettre une création

dynamique de contenu afin de pouvoir, le cas échéant, se lier facilement à une plateforme extérieure (Web Service, etc …).

Deux solutions différentes de gestion des données sont proposées. Une base de donnée de type SQLite et son utilisation classique par le biais de requêtes SQL ainsi que l'utilisation du Framework CoreData, l'outil de persistance des données fournit par Apple.

Une analyse et une utilisation des diverses possibilités d'interaction de l'Iphone a été effectuée. Elle comprend l'utilisation du micro, les possibilités tactiles de l'écran et les mouvements de l'appareil dans l'espace.

Lors du développement d'une application IPhone, Apple demande une validation de celle-ci avant sa mise à disposition sur l'Apple Store.

Ce projet étant un travail de recherche et d'étude des possibilités offertes par l'appareil pour notre thème, la validation et la mise à disposition n'est pas nécessaire. Nos applications seront disponibles à titre de démonstration pour le personnel de l'Institut.

Choix

Afin de répondre au mieux aux objectifs, deux applications principales et une optionnelle ont été développées.

IPos Simulator

Afin d'utiliser au maximum les possibilités d'interactions offertes par l'IPhone et toujours sur le modèle de l'asthme, un simulateur de ventolin a été développé.

Se basant sur le mode d'emploi de l'appareil, il permet d'effectuer toutes les opérations nécessaires à sa bonne utilisation.

IPos Quizz

Dans l'optique d'apprendre à connaître une maladie, un modèle de jeu de quizz a été développé. Il permet de mettre à disposition plusieurs séries de questions par niveau ou par thème.

Le modèle de donnée est assez souple afin de pouvoir supporter des questions sur n'importe quel sujet et n'importe quelle forme.

Dans un premier temps, des séries de questions simples sur les animaux seront mises en place afin de concentrer "le client" sur la forme plutôt que sur le fond. En effet, nous chercherons à le rendre attentif sur les possibilités de l'application plutôt que sur des questions spécifiques à une maladie.

Une série relative à l'asthme sera également créée afin de montrer une application du quizz à un domaine médical spécifique.

IPos Action

Au vu de l'avancement des travaux une application supplémentaire a été développée. Dans le cadre d'un traitement pour une maladie chronique, des plans d'action sont définis par le médecin du patient. Ceux-ci contiennent des zones d'état auquel le patient peut s'identifier par le biais de symptômes et accomplir des actions afin d'améliorer sa situation.

Une interface simple sera mise en place afin que le patient puisse facilement sélectionner son état et les actions pouvant contribuer à améliorer son état.

Un panneau de configuration sera mis à disposition du médecin afin qu'il puisse adapter les actions à son patient.

IPos Simulator

Introduction

Cette partie présente une analyse des besoins et un schéma de fonctionnement de l'application de Simulateur ainsi qu'une description détaillée des écrans et de leurs particularités.

Objectifs généraux

En suivant le thème de l'asthme, cette application permet un apprentissage de l'utilisation d'un ventolin. Elle s'adresse à toutes les personnes souffrant de cette maladie et nécessitant l'utilisation de cet appareil.

Par le biais des moyens d'interaction offerts par l'Iphone, l'apprenant est accompagné de manière ludique durant les diverses étapes nécessaires à une utilisation correcte de l'engin.

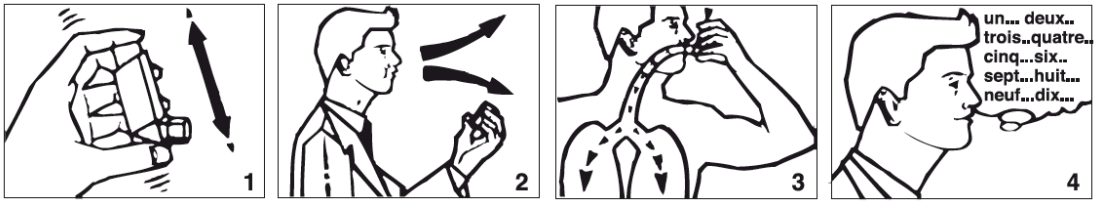
Objectifs technologiques

Cette application utilise au maximum les capacités interactives de l'Iphone. Elle propose une simulation des diverses étapes du processus d'utilisation d'un ventolin par le biais des canaux audio tactiles et de déplacement dans l'espace.

Pour cela, une récupération des gestes de l'utilisateur sur l'écran, de son interaction avec le micro et des mouvements effectués avec l'Iphone assure une simulation proche de la réalité.

Analyse des besoins

L'objectif de l'application de simulateur est de fournir un modèle d'apprentissage d'utilisation d'un ventolin respectant le protocole d'utilisation de celui-ci.



Pour ce faire, chaque étape décrite dans le mode d'emploi de l'appareil, en version complète sur le cd joint au dossier, a été reproduite en utilisant les moyens offerts par l'IPhone.

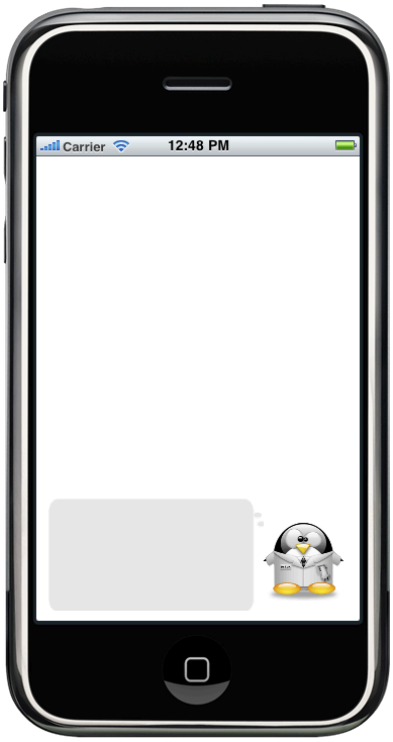
L'application reproduit donc les étapes suivantes :

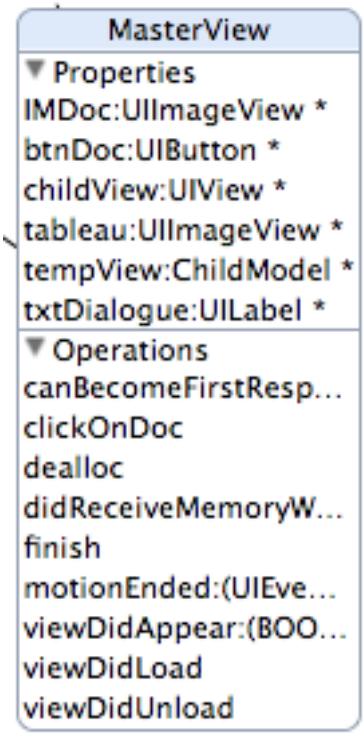
1. Mettre l'appareil dans la bonne position
2. Secouer énergiquement le ventolin
3. Vider ses poumons
4. Placer l'engin devant la bouche
5. Inspirant en appuyant sur le bouton supérieur
6. Retenir sa respiration durant 10 secondes

Fonctionnement

MasterView

Une vue principale a été mise en place, elle est chargée d'afficher les vues des différentes étapes, de gérer l'affichage des textes explicatifs et la navigation entre les vues.





Elle contient un label affichant les instructions à effectuer pour les différentes vues ainsi qu'un bouton (ici avec l'image d'un pingouin) qui permet à l'utilisateur de valider le changement d'écran.

Elle contient également une vue vide à laquelle seront attachées les différents écrans ainsi qu'une instance d'un ChildModel (explication ci-dessous) qui lui permettra de gérer ses différentes vues secondaires.

Opérations principales

Validation

Lorsque l'action demandée par la vue secondaire a été correctement exécutée, celle-ci appelle l'opération déléguée finish de la MasterView.

```
-(IBAction) finish{

    // animation de départ
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDelegate:self];
    [UIView setAnimationDuration:2];

    [txtDialogue setText:tempView.textDialogue];

    [UIView commitAnimations];

    tempView.correct = YES;
}
```

Celle-ci affiche le nouveau texte de dialogue et valide le ChildModel afin que l'utilisateur puisse passer à l'étape suivante.

Navigation

Quant l'action d'une vue secondaire a été validée et qu'elle a appelé l'opération déléguée finish (voir ci-dessus), l'utilisateur peut appuyer sur le bouton de navigation afin de passer à l'étape suivante et l'opération clickOnDoc est appelée.

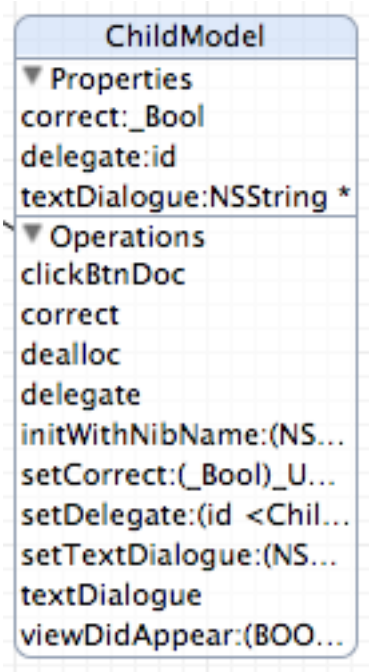
```
-(IBAction) clickOnDoc{
    if(tempView.correct){
    ChildModel * r = [tempView clickBtnDoc];

    r.delegate = self ;

    for (UIView *view in childView.subviews) {
        [view removeFromSuperview];
    }

    [childView addSubview:r.view];
    [childView setAlpha:0];
    [txtDialogue setAlpha:0];
    txtDialogue.text = r.textDialogue ;

    // animation de départ
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDelegate:self];
    [UIView setAnimationDuration:2];

    [txtDialogue setAlpha:1];
    [childView setAlpha:1];

    [UIView commitAnimations];

    tempView = r ;

    [tempView viewDidAppear:YES];
    }
}
```

Après un contrôle de la validité de l'action en cours, on appelle l'opération clickBtnDoc du ChildModel actuel, celui-ci renvoie le contrôleur de la prochaine vue à afficher.

La vue actuelle est alors effacée et la nouvelle affichée.

ChildModel

Cette classe contient les propriétés et les opérations communes à toutes les vues secondaires de l'application.



Elle contient une propriété contenant son statut, soit si l'action qu'elle demande a été effectuée ou non.

La propriété delegate contient la vue chargée de son affichage, soit ici la MasterView.

Le string textDialogue permet de stocker et de mettre à disposition le texte affiché selon l'état de validation du ChildModel.

Enfin l'opération clickBtnDoc doit être surdéfinie dans la classe fille et permet de donner le ChildModel devant succéder à celui-ci.

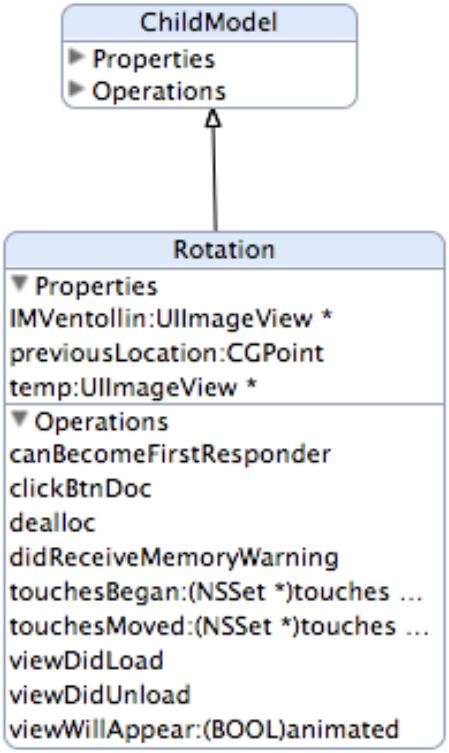
Exemple de surdéfinition de l'opération clickBtnDoc :

```
-(ChildModel *) clickBtnDoc{
    Rotation * r = [[Rotation alloc]initWithNibName:@"Rotation" bundle:nil];
    return r ;
}
```

Ecrans

Rotation

La première étape consiste à tourner le ventolin dans le bon sens. Par un processus de drag and drop l'utilisateur peut simplement tourner l'image dans le bon sens.



Elle contient un objet UIImage contenant l'image du ventolin ainsi qu'un CGPoint et une ImageView servant de variable temporaire afin de calculer le déplacement de l'objet (voir description d'opération).

Opérations principales

Nous allons utiliser les opérations de drag and drop vues précédemment dans l'application quizz.

Rotation

TouchesBegan

Lors de la pression sur l'écran, l'opération touchesBegan est appelée.

```
-(void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{

    UITouch * touch = [[event allTouches]anyObject];
    CGPoint location = [touch locationInView:touch.view];

    if (CGRectContainsPoint([IMVentollin frame], location)) {
        temp = IMVentollin ;
    }

    if(temp != nil ){
        previousLocation = location ;

    }

}
```

Si la pression s'effectue sur l'image du Ventolin (IMVentolin), nous copions l'image dans notre variable temporaire.

Cette opération peut paraître inutile mais elle est nécessaire afin que l'on sache si l'image est sélectionnée lors de l'opération touchesMoved. En effet, sans cette variable temporaire, peu importe l'endroit sélectionné, le mouvement s'effectuera.

Puis, nous sauvegardons la position de base dans notre variable previousLocation.

TouchesMoved

Lorsque la pression est suivie d'un déplacement, l'opération touchesMoved est appelée.

```
-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch * touch = [[event allTouches]anyObject];
    CGPoint location = [touch locationInView:touch.view];


    if(temp!=nil){


        CGFloat previousAngle = atan2(temp.center.x - previousLocation.x,
                                      temp.center.y - previousLocation.y) * 180 / M_PI;
        CGFloat currentAngle = atan2(temp.center.x - location.x,
                                     temp.center.y - location.y) * 180 / M_PI;

        CGFloat angleToRotate = previousAngle - currentAngle;
```

Après avoir contrôler si notre image a bien été sélectionnée par le biais de notre variable temp, nous calculons l'angle entre la position de base de l'image et le centre de l'image. Puis, nous calculons l'angle entre la position du doigt sur l'écran et le centre de l'image.

La différence nous donne l'angle de rotation à appliquer à notre image.

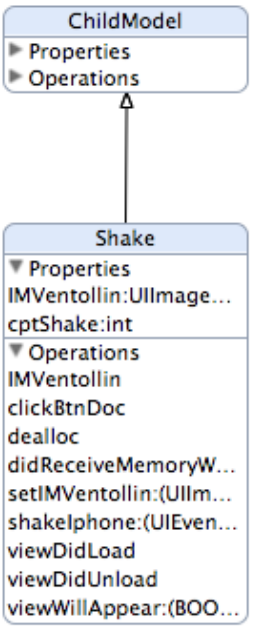
```
if (angleToRotate >= -185.0 && angleToRotate <= -175.0) {

    temp.transform = CGAffineTransformMakeRotation(degreesToRadians(180));
    temp = nil ;

    self.textDialogue = @"Super !!! Tape sur moi pour continuer ...";
    [self.delegate finish];
    return;
}
if (angleToRotate >= 175 && angleToRotate <= 185) {


    temp.transform = CGAffineTransformMakeRotation(degreesToRadians(180));
    temp = nil ;
    self.textDialogue = @"Super !!! Tape sur moi pour continuer ...";
    [self.delegate finish];

    return;
}

temp.transform = CGAffineTransformMakeRotation(degreesToRadians(angleToRotate));

[self.view bringSubviewToFront:temp];
```

Nous devons effectuer une rotation entre 175 et 185 degrés pour valider notre rotation. Nous testons donc si l'angle de rotation est entre ces deux valeurs et leur négatif.

Si c'est le cas, nous modifions notre texte de dialogue et appelons l'opération déléguée finish de notre MasterView. Sinon, nous appliquons l'angle de rotation à notre image.

Shake

La deuxième étape nous demande de secouer notre IPhone afin de mélanger le liquide de notre ventolin.



Le contrôleur Shake possède un compteur permettant de calculer le nombre de secousses effectuées ainsi qu'une UIImage affichant un ventolin.

Opérations principales

Seule la vue principale, ici la MasterView, peut recevoir l'événement de secousse. Elle contient donc l'opération motionEnded qui renvoit l'événement à notre Controlleur Shake par le biais de l'opération shakeIPhone.

```
- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event {

    if([tempView isMemberOfClass:[Shake class]]){
        Shake * temp = (Shake *) tempView ;
        [temp shakeIphone:event];
    }
}
```

ShakeIPhone

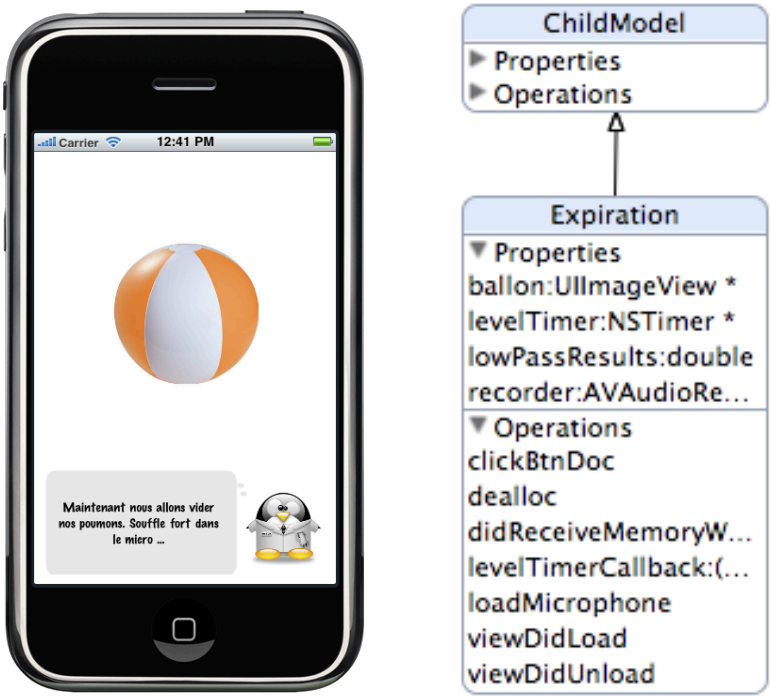
```
- (void) shakeIphone:(UIEvent *)event {
    if (event.type == UIEventSubtypeMotionShake) {

        if(cptShake>=2){

            // changement du texte
            self.textDialogue = @"Super !!! Tape pour continuer ...";

            [self.delegate finish];

            self.correct = FALSE;
            cptShake = 0 ;

        }
        else {

            if (cptShake>=1) {
                // changement du texte
                self.textDialogue = @"Encore une fois ...";

                [self.delegate finish];
                self.correct = FALSE ;

                cptShake++;
            }
            else {
                // changement du texte
                self.textDialogue = @"Encore ... ";

                [self.delegate finish];

                cptShake++;
            }
        }
    }
}
```

Ici nous calculons le nombre de secousses effectuées grâce à notre compteur. Pour chaque valeur, nous créons un nouveau dialogue, validons le changement à l'aide de l'opération déléguée finish de la MasterView et incrémentons le compteur.

La propriété correcte est mise à FALSE car lors de l'opération finish la MasterView valide l'action et la remplit donc à TRUE.

Expiration

La troisième étape nous demande de vider nos poumons en soufflant dans le micro de l'IPhone.



Le contrôleur Expiration possède une UIImage affichant un ballon. Celui-ci adapte sa taille à l'intensité du souffle produit.

Il contient également un enregistreur audio AVAudioRecorder, un NSTimer permettant de répéter l'opération d'enregistrement et un variable contenant le résultat de celui-ci.

Opérations principales

Chargement de l'enregistreur audio

Lors du chargement de la vue, nous créons un enregistreur audio par le biais de l'opération loadMicrophone.

```
-(void) loadMicrophone{

    NSURL *url = [NSURL fileURLWithPath:@"/dev/null"];

    NSDictionary *settings = [NSDictionary dictionaryWithObjectsAndKeys:
                              [NSNumber numberWithFloat: 44100.0],                AVSampleRateKey,
                              [NSNumber numberWithInt: kAudioFormatAppleLossless], AVFormatIDKey,
                              [NSNumber numberWithInt: 1],                        AVNumberOfChannelsKey,
                              [NSNumber numberWithInt: AVAudioQualityMax],        AVEncoderAudioQualityKey,
                              nil];

    NSError *error;

    recorder = [[AVAudioRecorder alloc] initWithURL:url settings:settings error:&error];

    if (recorder) {
        [recorder prepareToRecord];
        recorder.meteringEnabled = YES;
        [recorder record];
        levelTimer = [NSTimer scheduledTimerWithTimeInterval: 0.03 target: self
                                        selector: @selector(levelTimerCallback:) userInfo: nil repeats: YES];
    } else
        NSLog(@"erreur");
}
```

Ici, nous créons tout d'abord un chemin où notre enregistreur pourra sauvegarder ces données. Ensuite, nous créons un dictionnaire contenant les paramètres de notre enregistreur ainsi qu'une variable d'erreur. Finalement, nous créons notre enregistreur.

Nous enclenchons alors notre enregistreur puis créons un NSTimer qui exécutera chaque trois centième l'opération levelTimerCallBack, notre opération de traitement des données.

Traitement des données audio

Chaque trois centième, notre NSTimer exécute l'opération levelTimerCallBack qui traite les données enregistrées depuis le micro.

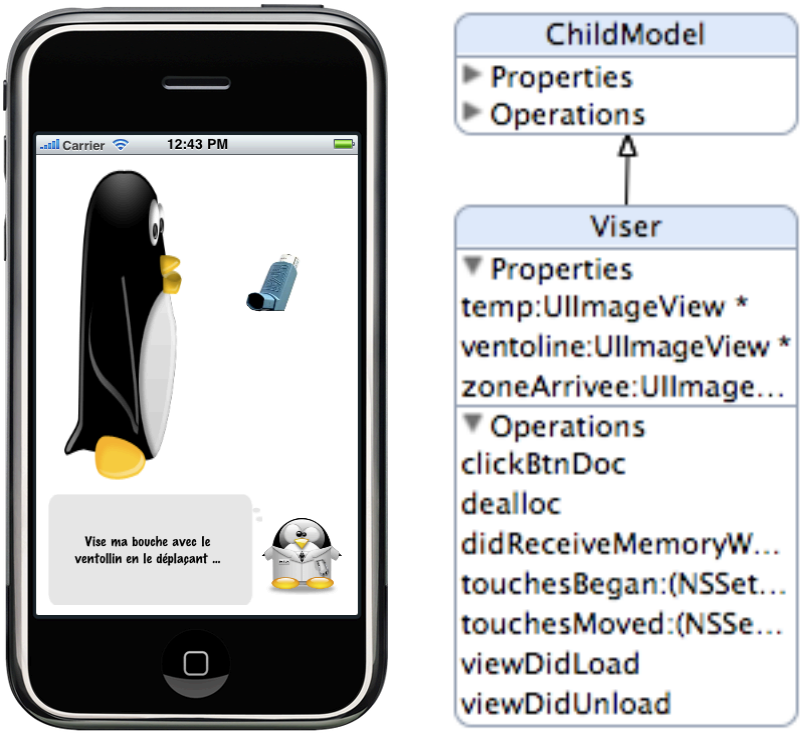
```
- (void)levelTimerCallback:(NSTimer *)timer {
    [recorder updateMeters];

    const double ALPHA = 0.05;
    double peakPowerForChannel = pow(10, (0.05 * [recorder peakPowerForChannel:0]));
    lowPassResults = ALPHA * peakPowerForChannel + (1.0 - ALPHA) * lowPassResults;

    CGRect r ;

    r.origin.x = 160 - ballon.frame.size.width/2;
    r.origin.y = 171 - ballon.frame.size.height/2;

    r.size.width = lowPassResults * 300;
    r.size.height = lowPassResults * 300;

    [ballon setFrame:r];

    if (lowPassResults >= 0.95){
        self.textDialogue = @"Génial !!! Retiens ta respiration et tape sur moi pour continuer ...";
        [self.delegate finish];
        self.correct = TRUE;
        [levelTimer invalidate];
    }
}
```

Ici, nous récupérons la valeur enregistrée dans notre variable lowPassResults. Ensuite, nous adaptons la taille du ballon par rapport à son intensité.

Finalement, nous contrôlons si la valeur dépasse 0.95, limite choisie pour valider l'action. Si c'est le cas, nous lançons l'opération déléguée finish de la MasterView et arrêtons le NSTimer.

Viser

La quatrième étape consiste à viser la bouche de notre pingouin avec le ventolin.



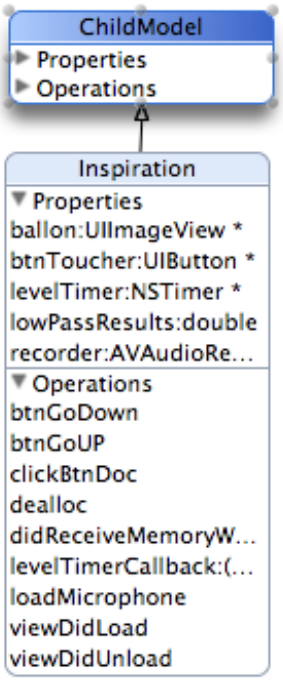
Ce contrôleur fonctionne exactement comme les questions Scrabble de notre application Quizz.

Il contient donc une UIImage affichant le ventolin, une zone d'Arrivée et une variable temporaire contenant l'objet à déplacer.

Les opérations touchesBegan, touchesMoved, touchesEnd sont implémentées afin de permettre le mouvement et la validation de la position du ventolin.

Inspiration

La cinquième étape nous demande d'inspirer dans le micro de l'Iphone tout en appuyant sur le bouton supérieur du ventolin.



Cette étape est implémentée de la même façon que l'étape d'expiration. Seul un détail change : L'enregistreur ne s'enclenche que lorsque nous appuyons sur le bouton supérieur du ventolin.

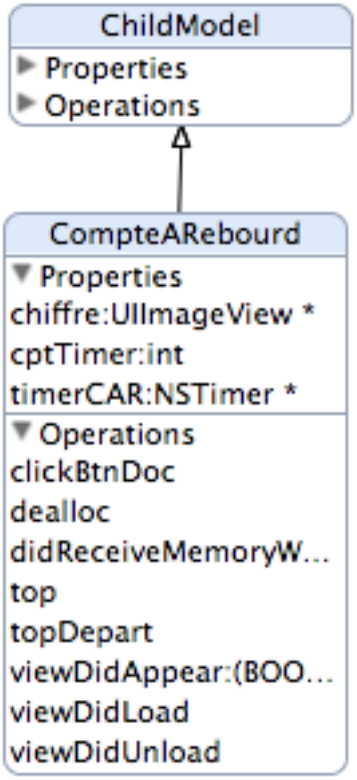
Elle possède donc un AVAudioRecorder, un NSTimer, un variable de résultat, une UIImage affichant un ballon et UIButton.

Les opérations loadMicrophone et levelTimerCallBack sont implémentées afin d'enregistrer et de traiter le flux audio.

L'UIImage affichant le ballon s'adapte à l'intensité du souffle en se dégonflant et la limite choisie est de 0.5.

Compte A Rebours

La dernière étape consiste à retenir son souffle durant 10 secondes. Un compte à rebours imagé apparaît sur l'écran de l'Iphone.



Le contrôleur CompteARebourd contient un NSTimer, un compteur contenant le nombre de secondes restantes et un UIImageView affichant les différentes images des chiffres.

Lors du chargement de la vue, nous initialisons un NSTimer qui lance à chaque seconde l'opération top. Celle-ci affiche l'image correspondante au chiffre contenu dans notre compteur que nous décrémentons par la suite.

Améliorations possibles

Cette application est une adaptation libre du mode d'emploi d'un ventolin. Une refonte de celle-ci en collaboration avec des professionnels est nécessaire afin de fournir un outil parfaitement adapté aux utilisateurs.

De nombreuses déclinaisons sur d'autres matériaux médicaux sont envisageables. C'est le cas par exemple, pour les étapes à effectuer par un diabétique pour une piqure d'insuline.

Difficultés rencontrées

Les difficultés de cette application ont surtout été d'ordre techniques. Chaque écran comprend des éléments d'acquisition différents, demandant une recherche et un apprentissage individuel.

Dans l'écran de rotation, le calcul des angles de rotation de l'image par rapport au doigt de l'utilisateur a demandé beaucoup de recherches de fonctions mathématiques et d'ajustements.

La récupération des données du micro pour les écrans d'expiration et d'inspiration a demandé, lui, des efforts de compréhension dans l'utilisation de l'AVAudioRecorder et de son paramétrage.

Finalement, au vu de l'importance d'un interface riche et attractive dans ce projet, l'enchaînement des écrans de manière fluide et agréable représente des heures d'ajustement, notamment sur la gestion des animations.

IPos Quizz

Introduction

Cette partie présente les objectifs généraux et technologiques, le fonctionnement général et les informations techniques de l'application de quizz.

Objectifs généraux

Cette application a pour but de permettre l'apprentissage de manière ludique. Sous la forme d'un quizz, elle va permettre à l'apprenant de mieux connaître sa maladie et de mieux la gérer.

Plusieurs types de questions sont développés afin de rendre son utilisation moins linéaire.

Elle permet d'effectuer des séries de questions différentes et de visualiser leurs résultats de manière simple.

Cette application étant destinée à montrer ses possibilités à des professionnels du domaine médical, des séries de questions portant sur des thèmes différents sont proposées. Afin de focaliser l'attention de ceux-ci sur la forme plutôt que sur le fond, deux séries de questions sur les animaux sont proposées. Une série plus spécifique, sur les voies respiratoires, est également mise à disposition, afin de démontrer une utilisation dans un domaine plus spécifique.

Objectifs technologiques

Sous la forme d'une application native, ce quizz ne demande aucune connexion extérieure. Les données (questions, séries, réponses) sont stockées sur une base de données de type SQLite.

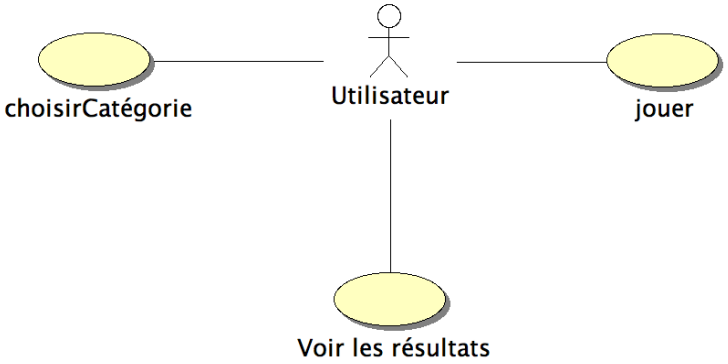
Une classe d'accès aux données gère la communication avec celle-ci par le biais de requêtes SQL.

Les séries de questions sont créées au démarrage du programme. De cette manière, l'application peut aisément être modifiée afin de se connecter à une plateforme extérieure lui fournissant des données et ainsi rendre son contenu plus dynamique.

En plus de questions à choix classiques, elle inclut un type de questions utilisant du drag and drop, assurant ainsi une utilisation moins monotone du quizz.

Analyse

Public cible : l'application s'adresse avant tout à des enfants de 8 à 12 ans atteints de maladies nécessitant une posologie régulière.



Choisir sa catégorie

Le quizz offre plusieurs séries de questions qui peuvent être aussi bien des niveaux de difficulté que des catégories de question. Cela permet de créer des modèles d'apprentissage pour tout âge et toute maladie.

Voir les résultats

Un affichage simple permet de voir les résultats de l'apprenant pour chaque série sous la forme de pourcentage.

Jouer

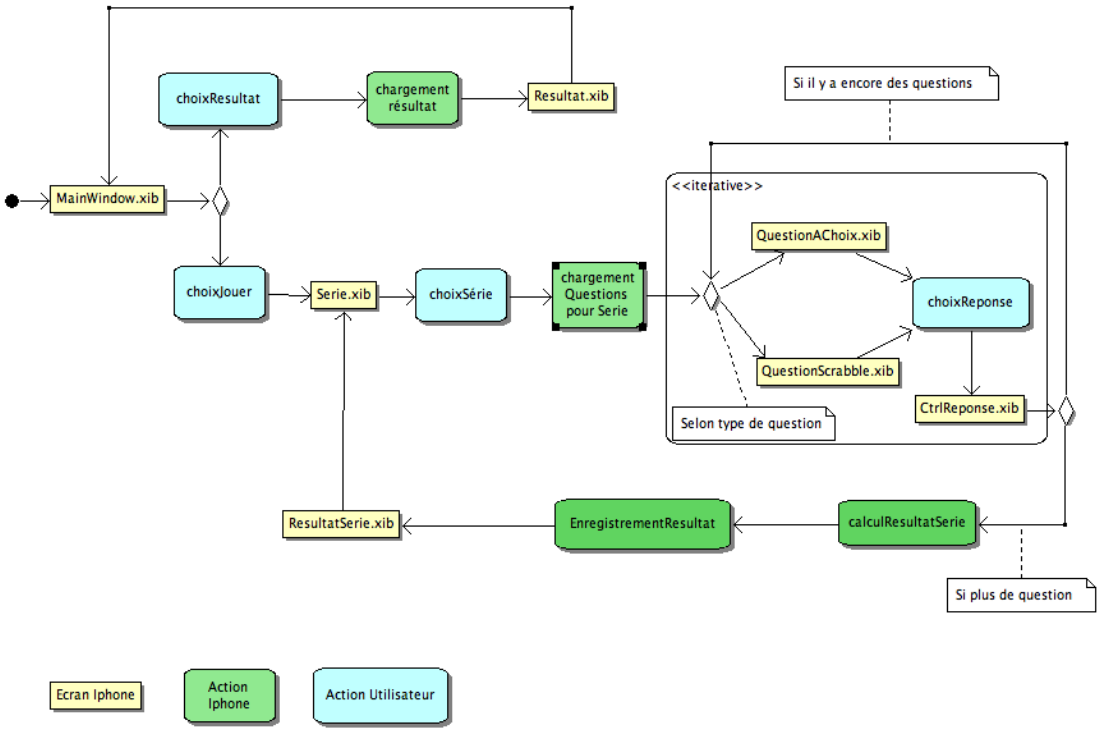
Après le choix d'une série, les questions se succèdent de manière dynamique jusqu'à l'arrivée à une page de résultat.

Fonctionnement général

Cette partie détaille le fonctionnement de l'application. Un schéma global, un détail des types de questions et de leur fonctionnement ainsi qu'un scénario d'utilisation sont présentés ci-dessous.

Schéma global

Le programme se déroule selon le fonctionnement suivant



Questions

Deux types de questions ont été développés dans le but de diversifier l'utilisation de l'application.

Questions à Choix

Les questions à choix sont assez classiques. L'utilisateur se voit proposer plusieurs réponses (entre deux et quatre propositions) et il doit simplement en choisir une.

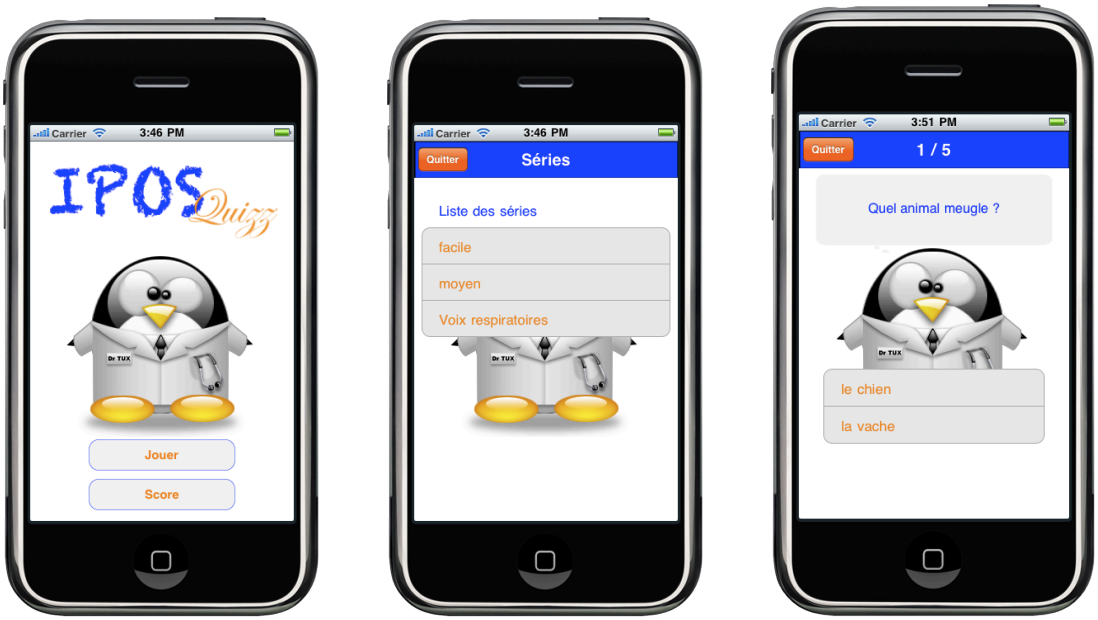
Questions Scrabble

Les questions scrabble proposent plusieurs lettres mélangées, l'utilisateur doit les glisser dans le champ de résultat dans le bon ordre afin de composer la réponse. La réponse est validée lorsque toutes les lettres proposées sont utilisées.

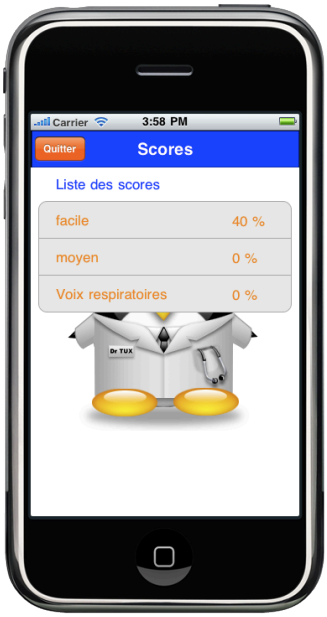
Pour chaque type de questions, lorsqu'une réponse a été sélectionnée, un écran de contrôle de la réponse apparaît. Il indique simplement si la réponse donnée est correcte ou non, et, le cas échéant, quelle était la bonne réponse.

Scénario d'utilisation

Le premier écran est le menu principal, l'utilisateur peut décider de jouer ou de consulter la liste des scores. En choisissant de jouer, il arrive sur l'écran de sélection des séries, où il peut choisir quelle catégorie il souhaite compléter. Puis, les questions s'enchaînent.



A la fin de la série de questions, un écran de résultats apparaît. L'utilisateur peut voir son score puis revenir à l'écran principal et ensuite regarder le tableau récapitulatif de bilan.

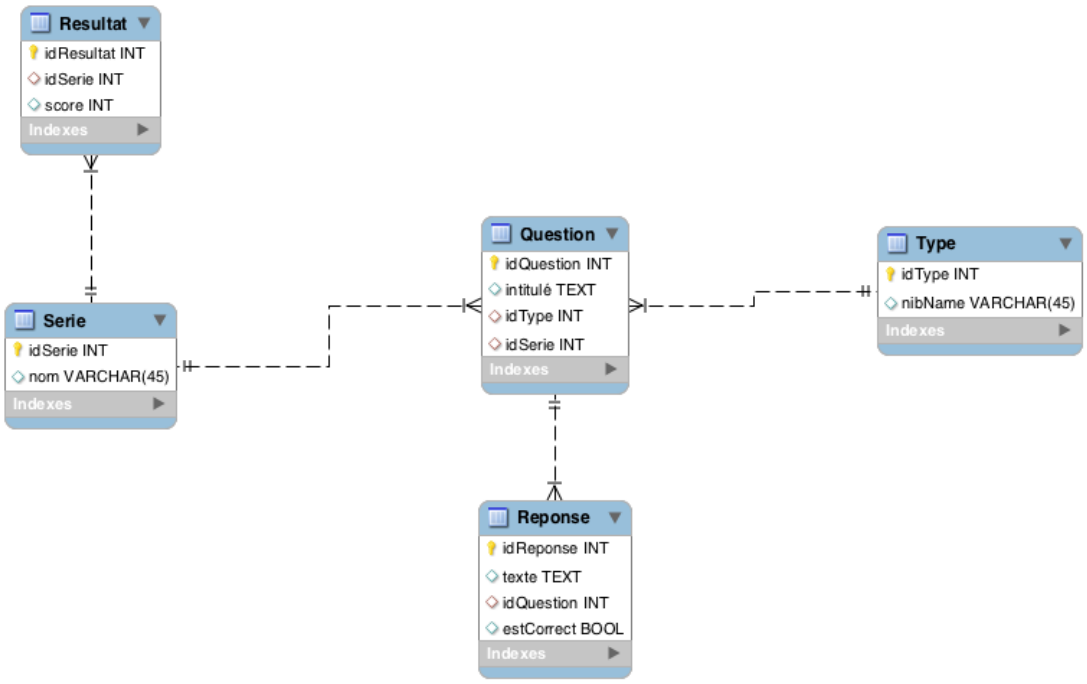


Informations techniques

Ici nous aborderons la partie technique du projet. Nous passerons en revue la modélisation et la documentation complète du développement.

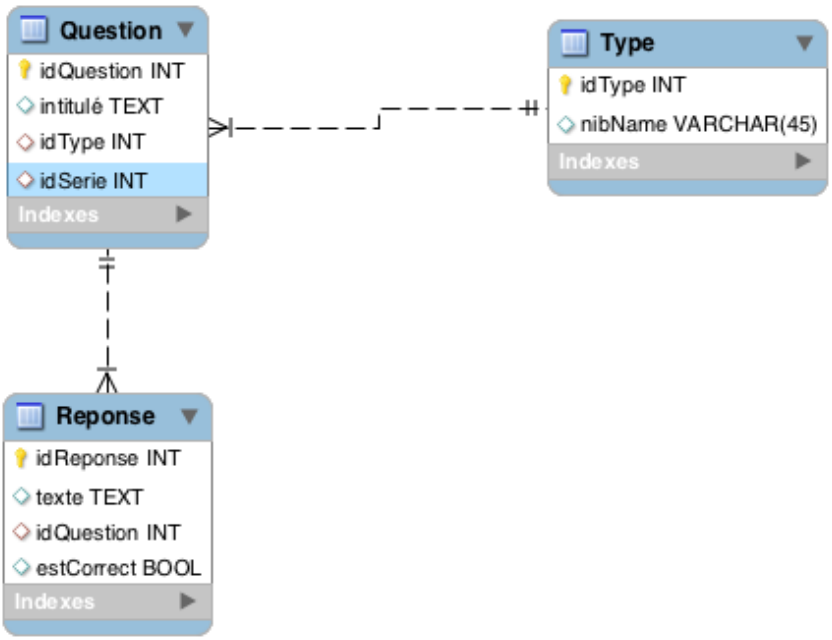
Modélisation

Après une analyse des besoins de l'application, des données nécessitant une persistance et des extensions futures possibles, le modèle de données suivant vous est présenté, ainsi qu'une analyse détaillée de celui-ci.



Question

La question est le point central de l'application. Elle doit répondre à certains critères afin de rester le plus souple possible et de permettre n'importe quel affichage.



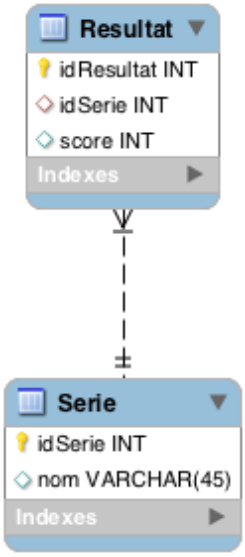
Chaque question contient son intitulé ainsi qu'un type et un identifiant de série.

Le type permet d'identifier le mode d'affichage de la question (A Choix ou Scrabble). Le chant "NibName" contient le nom de la vue à charger pour ce type de question.

Une ou plusieurs réponses peuvent être liées à une question selon son type. Une réponse contient son texte ainsi qu'un champ booléen indiquant s'il s'agit d'une réponse correcte ou incorrecte.

Série

Les séries sont les catégories de questions, elles organisent les questions en enchaînement selon des critères de difficulté ou de genre.



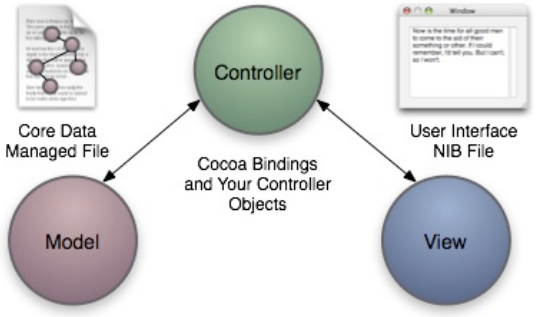
Chaque série a son propre nom la définissant et est liée à un résultat stocké en pourcentage de réussite.

Développement

La plate-forme de développement fournie par Apple se base sur le modèle MVC (ModelViewControler) qui vise à séparer l'affichage et le modèle de données et le fonctionnement général de l'application.

Cette partie présente les trois couches de ce projet :

- Les modèles (classes métiers et accès aux données)
- Les vues (l'affichage)
- Les contrôleurs (gestion des événements et des modifications)



Modèles

Pour chaque table du modèle de données, une classe a été créée selon un squelette classique en objectif –C.

- un constructeur simple
- un constructeur d'initialisation
- un destructeur
- les getter et setter nécessaires à l'accès aux données

Elles incluent également les opérations nécessaires au fonctionnement du programme.

```
//
//  Resultat.h
//  IPosQuizz
//
//  Created by francois on 04.06.10.
//  Copyright 2010 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>


@interface Resultat : NSObject {

    @private
    NSNumber * idResultat ;
    NSNumber * score ;

}

@property (readwrite, retain, nonatomic) NSNumber * idResultat ;
@property (readwrite, retain, nonatomic) NSNumber * score ;

- (Resultat *) initWithValue : (NSNumber*) idR :(NSNumber*) s;
-(void) dealloc ;

@end
```

Dans l'application de quizz les modèles et l'accès aux données ont été créés manuellement.

Au vu du nombre peu important de requêtes à effectuer, une classe d'accès aux données a été créée. Appelée dans le programme, elle se charge de générer les objets nécessaires depuis la base de donnée.

Vues et contrôleurs

Vues

Les vues sont la partie visuelle de l'application, chaque vue possède un fichier .xib qui contient les informations graphiques générales de l'écran à afficher. Elle est directement liée à son contrôleur et est créée par celui-ci.

Contrôleur

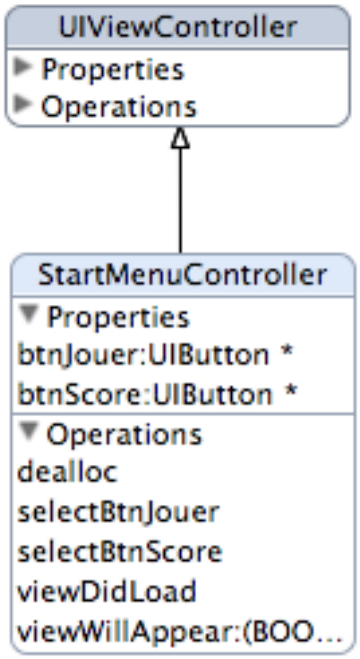
Les contrôleurs sont quant à eux le cœur de l'application. C'est eux qui vont gérer les modèles et indiquer de quelle manière les afficher dans la vue.

Le contrôleur hérite de la classe UIViewController qui contient toutes les propriétés et opérations de base fournit par Apple afin de contrôler une vue.

Voici un exemple avec une vue classique, celle du menu principal et de son contrôleur. On voit ici qu'il possède comme propriété, les deux boutons permettant une interaction.

Il possède également deux opérations "select" qui permettent de définir l'action à effectuer en cas de sélection de l'un des deux boutons.

Les opérations viewDidLoad et viewWillAppear sont quant à elles des surdéfinition des opérations de base d'un UIViewController et permettent d'effectuer des modifications lors du chargement de la vue.



Navigation

Window et NavigationController

Nous avons vu ci-dessus l'architecture d'une vue et de son contrôleur. Nous allons maintenant voir de quelle manière s'effectue la navigation entre les vues et la gestion de celle –ci.

Lors du chargement de l'application, une fenêtre principale est créée elle affichera les différentes vues. A l'intérieur du contrôleur de cette fenêtre, nous allons créer un NavigationController. Cette classe, mise à disposition, nous permet de gérer facilement la navigation entre nos différentes vues.

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    StartMenuController * sm = [[StartMenuController alloc]initWithNibName:@"StartMenu" bundle:nil];
    UINavigationController * nav = [[UINavigationController alloc]initWithRootViewController:sm];
    [window addSubview:nav.view];
    [window makeKeyAndVisible];
}
```

Nous voyons ici qu'à la fin du chargement de l'application, nous créons une instance de notre classe StartMenuController en lui passant le nom de sa vue "StartMenu".

Nous créons, ensuite, un NavigationController en lui passant comme fenêtre racine notre vue principale.

Finalement, nous ajoutons notre vue à la fenêtre afin qu'elle s'affiche au départ de notre application.

Changement de vue

Nous avons vu tout à l'heure que notre classe StartMenuController contenait une opération selectBtnJouer qui doit lancer la vue de sélection des séries. Elle contient le code suivant :

```
- (IBAction) selectBtnJouer {
    ListeSerieController * lsc = [[ListeSerieController alloc] initWithNibName:@"ListeSerie" bundle:nil];
    [self.navigationController pushViewController:lsc animated:YES ];
}
```

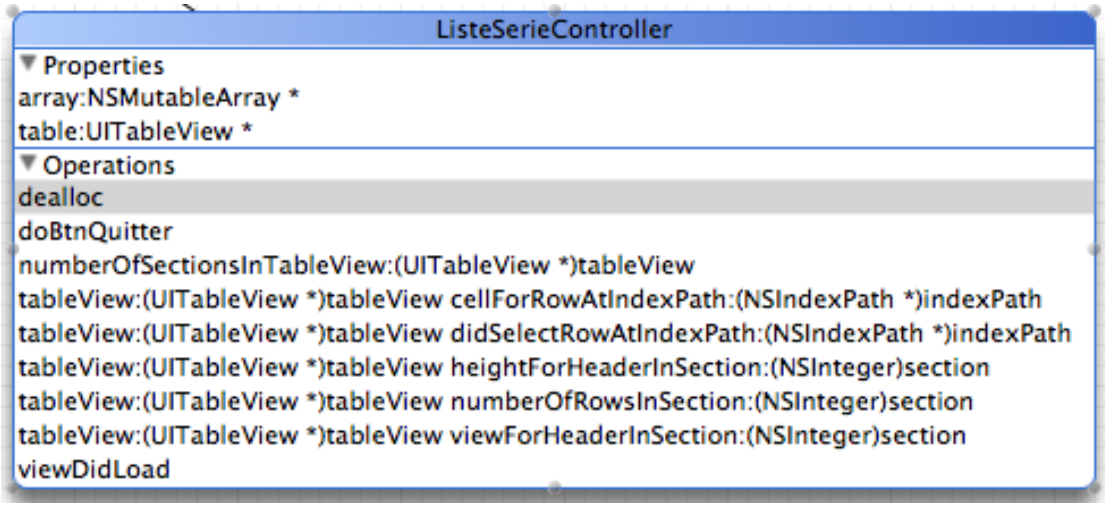
Nous créons ici le controller suivant et demandons à notre NaviguationController de le "pousser" comme vue à afficher. La vue de ce controller apparaît avec l'animation classique des applications IPhone.

Contrôleurs spécifiques

TableViewController

Dans la vue ci-dessous, ListeSerieController, nous voyons l'apparition d'une liste de données. Cet objet est un descendant d'une vue classique auquel nous allons rattacher le contrôleur de notre vue par les propriétés datasource et delegate. Ainsi nous pourrons mettre le code spécifique de notre tableau directement dans le même contrôleur.

Observons notre contrôleur.



Notre contrôleur contient un MutableArray qui est le tableau contenant les éléments que l'on souhaite afficher et qui est rempli lors du chargement de la vue. Il contient également notre TableView afin de pouvoir la gérer.

Les opérations suivantes sont implémentées afin d'assurer le bon fonctionnement du tableau.

NumberOfSectionsInTableView

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{
    return 1;
}
```

On renvoie le nombre de sections de la liste. Ici, nous en avons une seule.

NumberOfRowsInSection

```
// Customize the number of rows in the table view.
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {

    return [array count] ;

}
```

On renvoie le nombre de lignes à créer dans la section, soit la taille de notre tableau.

HeightForHeaderInSection

```
// hauteur du titre de la table view
- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteger)section {

    return 40;
}
```

Ici nous renvoyons la hauteur désirée de notre vue de titre.

ViewForHeaderInSection

```
// création du titre de la table view
- (UIView *) tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section  {

     UIView *headerView = [[[UIView alloc]
                            initWithFrame:CGRectMake(10, 0, tableView.bounds.size.width, 40)]
                           autorelease];

    UILabel *label = [[[UILabel alloc]
                       initWithFrame:CGRectMake(30, 10, tableView.bounds.size.width - 10, 20)]
                      autorelease];

    label.text = @"Liste des séries";
    label.textColor = [UIColor blueColor];
    label.backgroundColor = [UIColor whiteColor];
    [headerView addSubview:label];

    return headerView ;
}
```

Ici nous créons la vue contenant le titre de la liste et la renvoyons.

CellForRowAtIndexPath

```
// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    int storyIndex = [indexPath indexAtPosition:[indexPath length]-1];

    static NSString *CellIdentifier = @"CellIndent";

    CelluleListeSerieController *cell = (CelluleListeSerieController *)
        [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

    if (cell == nil) {
        UIViewController *c = [[UIViewController alloc]
                initWithNibName:@"CelluleListeSerie" bundle:nil];
        cell = (CelluleListeSerieController *)c.view;

    }

    Serie * q = (Serie*) [array objectAtIndex:storyIndex] ;

    cell.lblSerie.text = q.nom ;

    return cell;

}
```

Ici nous créons une cellule pour chaque élément du tableau. Cette cellule est un objet qui hérite de UITableViewCell et possède sa propre vue et son propre contrôleur. Nous pouvons ainsi personnaliser le contenu de nos cellules.

DidSelectRowAtIndexPath

```
// en cas de selection
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

    Serie * s ;

    int storyIndex = [indexPath indexAtPosition:[indexPath length]-1];

    s = [array objectAtIndex:storyIndex]  ;

    SqlManager * sql = [[SqlManager alloc]init];

    s = [sql getCompleteSerie:s] ;

    [sql release];

    s.resultat.score = [NSNumber numberWithInt:0];


    //premiere question
    Question * temp = [s.questions objectAtIndex:0] ;

    if([temp.type.nibName isEqualToString:@"QuestionAChoix" ]){

        QuestionAChoixController * c = [[QuestionAChoixController alloc]initWithNibNameandQuestion:
                                        temp.type.nibName bundle:nil serie:s cpt:0];

        [self.navigationController pushViewController:c animated:YES];

        [c release];
        c = nil;

    }
    else{
        if([temp.type.nibName isEqualToString:@"QuestionScrabble" ]){

            QuestionScrabbleController * c = [[QuestionScrabbleController alloc]initWithNibNameandQuestion:
                                              temp.type.nibName bundle:nil serie:s cpt:0];

            [self.navigationController pushViewController:c animated:YES];

            [c release];
            c = nil;
        }
    }
}
```

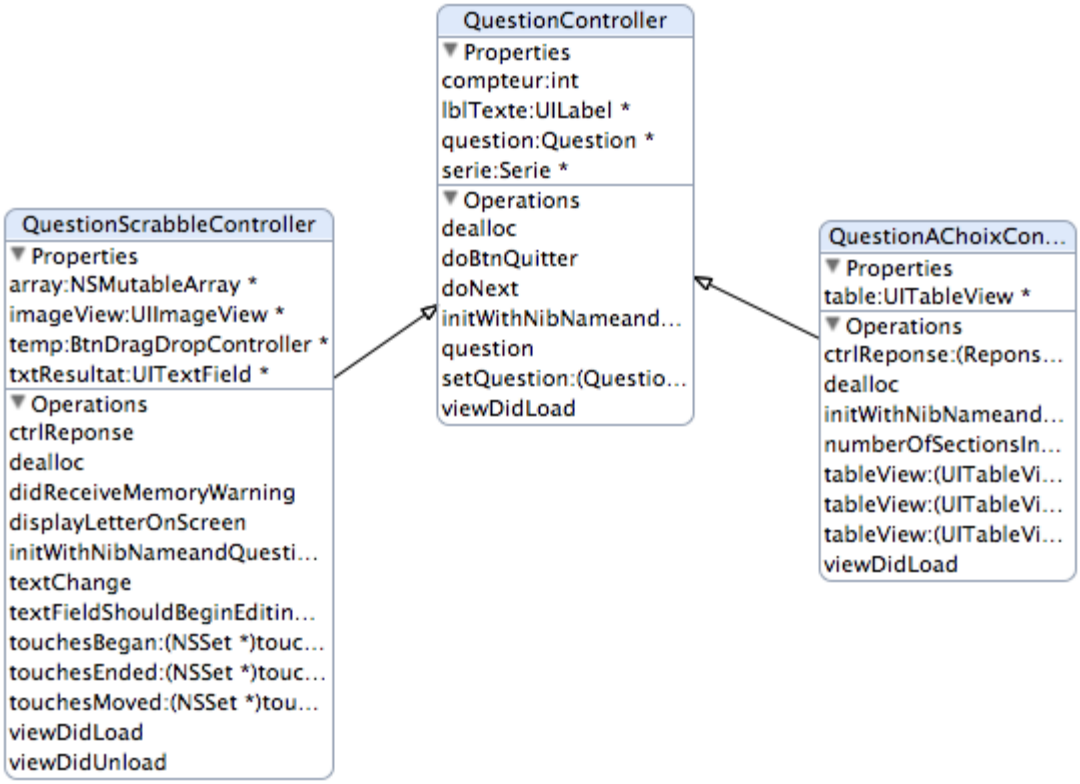
Ici nous allons définir l'action à effectuer lorsque l'on sélectionne une ligne de la liste. Dans notre vue cela lancera la première question de la série choisie. On récupère donc l'index de la ligne sélectionnée. Ensuite, on charge un objet Série contenant les questions de celle-ci.

On regarde ensuite le type de la première question et créons le contrôleur nécessaire. Nous "poussons" ensuite la vue de celui-ci et la première question s'affiche.

QuestionController

Un contrôleur de questions contient la série à laquelle il appartient, un compteur afin de savoir où il se trouve dans la série et un label affichant la question. Il doit pouvoir passer à la question suivante et quitter la série.

Ces propriétés et opérations ne changent pas, peu importe le type de questions. Elles font donc partie de la classe QuestionController de laquelle héritent les différents types de questions.



Lors de la validation de la réponse par le contrôleur, l'opération doNext est appelée. Elle permet grâce au compteur et à l'objet Série de charger et d'afficher la prochaine question ou, si la série est terminée,de créer l'écran de résultats.

DoNext

```
-(void) doNext{

    compteur++;

    if(compteur < [serie.questions count]){

        //prochaine question
        Question * temp = [serie.questions objectAtIndex:compteur] ;

        if([temp.type.nibName isEqualToString:@"QuestionAChoix"]){

            QuestionAChoixController * c = [[QuestionAChoixController alloc]initWithNibNameandQuestion:
                                            temp.type.nibName bundle:nil serie:serie cpt:compteur];

            [self dismissModalViewControllerAnimated:NO];

            [self.navigationController pushViewController:c animated:YES];


        }
        else{
            if([temp.type.nibName isEqualToString:@"QuestionScrabble" ]){

                QuestionScrabbleController * c = [[QuestionScrabbleController alloc]initWithNibNameandQuestion:
                                                  temp.type.nibName bundle:nil serie:serie cpt:compteur];


                [self dismissModalViewControllerAnimated:NO];
                [self.navigationController pushViewController:c animated:YES];

            }
        }


    }
    else{

        ResultatSerie * r = [[ResultatSerie alloc]initWithNibName:@"ResultatSerie" bundle:nil serie:serie];

        [self dismissModalViewControllerAnimated:NO];

        [self.navigationController pushViewController:r animated:YES];

    }
}
```

Ici, nous incrémentons le compteur de position dans la série. Après contrôle de la longueur de la série, nous créons la question suivante selon son type. Si la série est terminée, nous créons la vue de résultats de la série et son contrôleur ResultatSerie.

QuestionAChoixController

Le contrôleur des questions à choix hérite de Question Controlleur. Il s'agit là d'une simple vue avec une TableView comme montré lors de la présentation de la liste des séries. Elle implémente les opérations nécessaires au fonctionnement de la TableView et une opération de contrôle de la réponse CtrlReponse.

CtrlReponse

```
-(void) ctrlReponse:(Reponse *) r{

    CtrlReponse * mod;

    if(r.estCorrect){

        mod = [[CtrlReponse alloc]initWithNibName:@"CtrlReponse" bundle:nil lbl1:nil type:0 quest:self];
        //NSLog([NSString stringWithFormat:@"%@", self.serie.resultat.score]);
        serie.resultat.score = [NSNumber numberWithInt:[serie.resultat.score intValue] + 1] ;

    }
    else {

        Reponse * temp;

        for(int i=0;i<[question.reponses count];i++){
            temp = [question.reponses objectAtIndex:i];
            if(temp.estCorrect){

                r = [self.question.reponses objectAtIndex:i];
            }
        }


        mod = [[CtrlReponse alloc]initWithNibName:@"CtrlReponse" bundle:nil lbl1:r.texte type:1 quest:self];

    }

    mod.modalTransitionStyle = UIModalTransitionStyleFlipHorizontal;
    [self presentModalViewController:mod animated:YES];
}
```

Grâce à la réponse sélectionnée dans le tableau, on contrôle si elle est correcte ou non. Ensuite, une vue et un contrôleur CtrlReponse sont instanciés. La vue consiste en un écran qui affiche si la réponse est correcte ou non et, le cas échéant la réponse exacte.

Pour la mettre à l'écran, nous utilisons le ModalViewControler inclu dans la classe ViewController et qui permet d'afficher une vue directement dépendante de celle-ci.

C'est cette vue qui appellera l'opération doNext de QuestionController et fera ainsi passer à la question suivante.

QuestionScrabbleController

La classe QuestionScrabbleController hérite également de la classe QuestionController. Elle possède les propriétés suivantes :

- un champ texte de résultat
- une zone d'accroche lors du drag and drop des lettres
- un tableau des BtnDragDrop dispatcher sur l'écran
- une instance de BtnDragDrop permettant de connaître la lettre qui est en train d'être déplacée.
  La classe BtnDragDrop hérite de la classe UIButton. Elle représente un bouton contenant une lettre et les opérations nécessaires à son fonctionnement.

Chargement des lettres sur l'écran

```
-(void) displayLetterOnScreen{
    Reponse * rep = [question.reponses objectAtIndex:0];

    NSString * repValue = rep.texte ;

    int s = [repValue length] ;


    for(NSUInteger i=0;i<s;i++){
        CGRect r ;

        bool tempBool = FALSE;
        while (!tempBool) {

            // randomize the place on the screen
            int k = rand() % 6;
            int j = rand() % 4;

            CGPoint p ;

            p.x = 20 + k * 47 ;
            p.y = 100 + j*50 ;

            CGSize s ;

            s.width = 40 ;
            s.height = 40 ;

            r.size = s ;
            r.origin = p ;

            tempBool = TRUE;

            for(int m=0;m<[array count];m++){

                BtnDragDropController * tempImView = [array objectAtIndex:m];

                if(r.origin.x == tempImView.frame.origin.x ){
                    if(r.origin.y == tempImView.frame.origin.y){
                        tempBool = FALSE;
                    }
                }
            }
        }

        unichar tempValue = [repValue characterAtIndex:i];
        BtnDragDropController * t = [[BtnDragDropController alloc]initWithFrame:r
                                        lbl:[NSString stringWithFormat:@"%C", tempValue]];

        [array addObject:t];
        [self.view addSubview:t];
        tempBool = FALSE;
    }
}
```

L'opération displayLetterOnScreen permet de répartir sur l'écran les différentes lettres composant le résultat de la question scrabble.

Pour chaque lettre, nous allons créer une instance de BtnDragDrop qui contiendra la lettre. Afin de la positionner, nous allons utiliser un chiffre aléatoire et calculer une position dans la surface d'affichage.

Après chaque création de coordonnée, avant de créer l'objet proprement dit, nous contrôlons qu'aucun autre objet ne chevauche celui-ci. Lorsque les coordonnées sont correctes, nous créons l'objet et l'ajoutons à notre vue et à notre tableau de BtnDragDrop.

Drag and Drop

Afin de pouvoir effectuer les Drag and Drop de nos BtnDragDrop, trois opérations sont à implémenter dans notre vue.

```
-(void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event

    UITouch * touch = [[event allTouches]anyObject];
    CGPoint location = [touch locationInView:touch.view];

    for (int i=0; i<[array count]; i++) {

        BtnDragDropController * cloud = [array objectAtIndex:i];

        if (CGRectContainsPoint([cloud frame], location)) {
            temp = cloud ;
        }

    }

    if(temp!=nil){
        temp.center = location;
        [self.view bringSubviewToFront:temp];
    }

}
```

Cette opération est appelée lorsque une pression est exercée sur l'écran. Nous calculons les coordonnées de la pression et les comparons à celle de nos objets. Si elle correspond à l'un de ceux-ci, nous le sauvegardons dans notre variable temp.

```
-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch * touch = [[event allTouches]anyObject];
    CGPoint location = [touch locationInView:touch.view];

    if(temp!=nil){
        temp.center = location;
        [self.view bringSubviewToFront:temp];
    }

}
```

La deuxième opération est appelée lors d'une pression et d'un déplacement. Ici, nous récupérons simplement notre temp et lui donnons la position de la pression. Ainsi, notre BtnDragDrop suit notre doigt.

```
-(void) touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event{

    UITouch * touch = [[event allTouches]anyObject];
    CGPoint location = [touch locationInView:touch.view];

        if (CGRectContainsPoint([imageView frame], location)) {
            if(temp!=nil){
            txtResultat.text = [NSString stringWithFormat:@"%@%@",txtResultat.text, temp.lblTexte.text] ;
            temp.hidden = YES;
            [array removeObject:temp];
            temp = nil;

            }
        }
        else {
            if(temp!=nil){
                temp.center = location;
                [self.view bringSubviewToFront:temp];
            }
        }

    if([array count] ==0){

        [self ctrlReponse];

    }

}
```

Enfin, la dernière opération est appelée lors du relâchement d'une pression. Ici, nous calculons la position du relâchement de pression. Si elle se trouve dans notre zone d'accroche, nous concaténons notre lettre au texte de notre champ de résultats et effaçons notre BtnDragDrop de notre tableau.

Ainsi, à la fin de l'événement, nous contrôlons si tous les BtnDragDrop ont été utilisés. Si c'est le cas, nous lançons l'opération de contrôle de la réponse.

Améliorations possibles

L'architecture de l'application permet de nombreuses extensions, en commençant bien sûr par le développement de nouveaux types de question.

L'idée d'un accès via Web Service à une base de données centrale de séries de questions est également intéressante. Elle permettrait à l'enseignant de créer ses propres séries, de récupérer les résultats des élèves et ainsi d'avoir un suivi des leçons effectuées.

Un projet similaire a été développé pour la plateforme CyberLearn dans le cadre d'un travail de diplôme. Elle proposait différent type de questions relatives à l'apprentissage de la programmation. Une adaptation de ce quizz pour accéder à ces questions pourrait être un autre vecteur de diffusion de cette offre.

Difficultés rencontrées

Cette application est la première à avoir été développée. Ayant déjà réalisé une application Iphone dans le cadre d'un projet, je possédais déjà des connaissances dans l'utilisation des objets standards du SDK d'Apple.

Cependant, au vu du besoin d'interfaces attractives et adaptées à des enfants, une transformation du graphisme de ceux-ci était nécessaire.

L'apprentissage des différentes méthodes de transformation de ceux-ci a donc été la plus grande difficulté pour cette partie du travail de diplôme.

IPos ActionPlan

Introduction

L'application de plan d'action est une partie optionnelle, ajoutée au vu du bon avancement de la réalisation de mon travail de diplôme. Initialement, une plateforme de distribution de questions pour la partie quizz était prévue.

Après discussion avec Mme Glassey, notre choix s'est porté sur cette application de plan d'action.

Objectifs généraux

Dans le cadre d'un traitement pour l'asthme, les médecins fournissent aux patients un plan d'action. Celui-ci leur permet d'évaluer leur état et leur donne des actions à effectuer afin de gérer leur maladie au mieux.

Cette application fournit une base d'interface d'un plan d'action ainsi que la possibilité de modifier les actions pour le médecin.

En effet, dans le cadre d'un plan d'action, le médecin et le patient remplissent le plan d'action ensemble afin d'assurer une bonne compréhension de celui-ci.

Une interface de configuration est implémentée et permet de manière rapide de lier des actions aux divers états possibles du patient.

L'application est inspirée d'un plan d'action existant. Cependant, les actions et les traitements proposés ont été créés pour les besoins de l'application et ne correspondent pas aux besoins réels d'un patient.

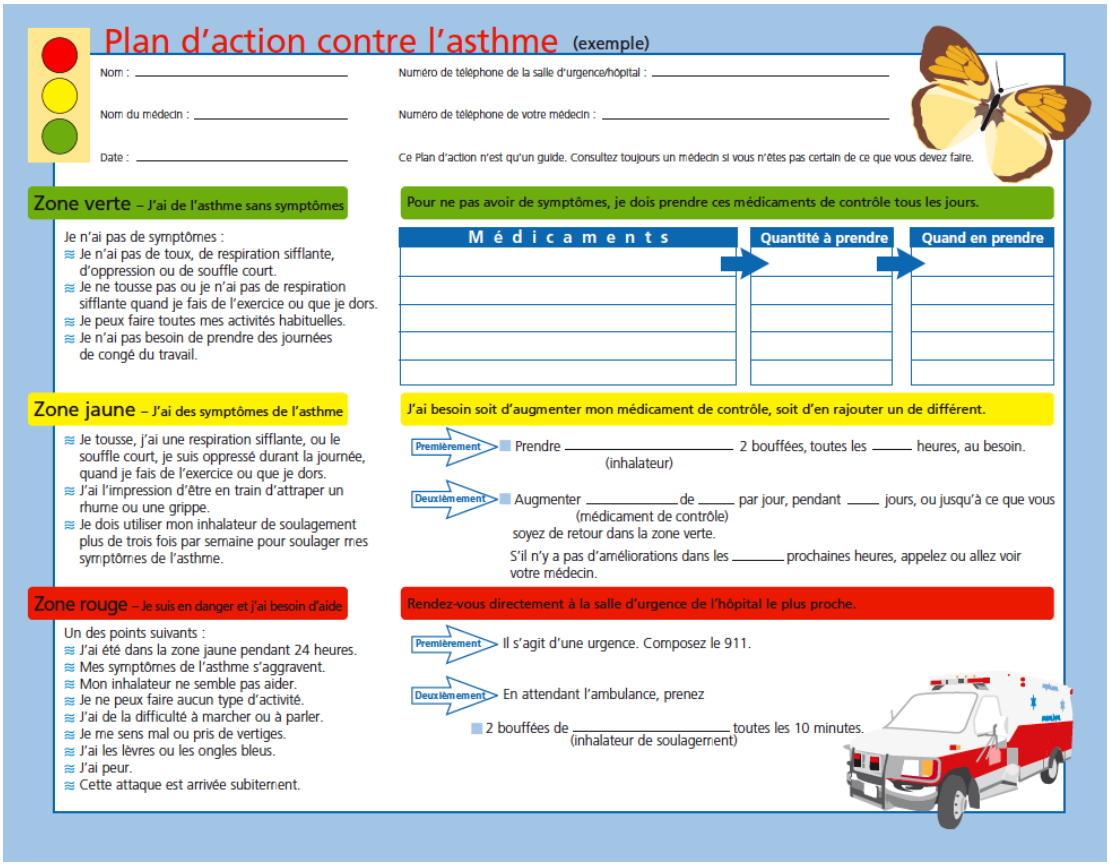
Objectifs technologiques

Comme l'application de quizz, le plan d'action est une application native. Les données sont créées lors du premier lancement de l'application et peuvent ensuite être modifiées. Ceci permettra, lors d'un développement futur, d'utiliser une plateforme externe (Web Service etc…) pour acquérir les données de base et ainsi assurer une adaptabilité maximum à l'application.

La gestion des données persistantes est effectuée grâce au Framework CoreData, fourni par Apple. Celui-ci permet un stockage automatique des données et une interface graphique de création des modèles. La création d'une base de données et d'une classe d'accès aux données n'est donc pas nécessaire.

Analyse

Voici le plan d'action qui a été utilisé comme base de travail :



Plan d'action contre l'asthme (exemple)

Nom : _____ Numéro de téléphone de la salle d'urgence/hôpital : _____

Nom du médecin : _____ Numéro de téléphone de votre médecin : _____

Date : _____ Ce Plan d'action n'est qu'un guide. Consultez toujours un médecin si vous n'êtes pas certain de ce que vous devez faire.

Zone verte – J'ai de l'asthme sans symptômes

Je n'ai pas de symptômes :
- Je n'ai pas de toux, de respiration sifflante, d'oppression ou de souffle court.
- Je ne tousse pas ou je n'ai pas de respiration sifflante quand je fais de l'exercice ou que je dors.
- Je peux faire toutes mes activités habituelles.
- Je n'ai pas besoin de prendre des journées de congé du travail.

Pour ne pas avoir de symptômes, je dois prendre ces médicaments de contrôle tous les jours.

Médicaments	Quantité à prendre	Quand en prendre

Zone jaune – J'ai des symptômes de l'asthme

- Je tousse, j'ai une respiration sifflante, ou le souffle court, je suis oppressé durant la journée, quand je fais de l'exercice ou que je dors.
- J'ai l'impression d'être en train d'attraper un rhume ou une grippe.
- Je dois utiliser mon inhalateur de soulagement plus de trois fois par semaine pour soulager mes symptômes de l'asthme.

J'ai besoin soit d'augmenter mon médicament de contrôle, soit d'en rajouter un de différent.

Premièrement : Prendre _____ (inhalateur) 2 bouffées, toutes les ____ heures, au besoin.

Deuxièmement : Augmenter _____ (médicament de contrôle) de ____ par jour, pendant ____ jours, ou jusqu'à ce que vous soyez de retour dans la zone verte.

S'il n'y a pas d'améliorations dans les _____ prochaines heures, appelez ou allez voir votre médecin.

Zone rouge – Je suis en danger et j'ai besoin d'aide

Un des points suivants :
- J'ai été dans la zone jaune pendant 24 heures.
- Mes symptômes de l'asthme s'aggravent.
- Mon inhalateur ne semble pas aider.
- Je ne peux faire aucun type d'activité.
- J'ai de la difficulté à marcher ou à parler.
- Je me sens mal ou pris de vertiges.
- J'ai les lèvres ou les ongles bleus.
- J'ai peur.
- Cette attaque est arrivée subitement.

Rendez-vous directement à la salle d'urgence de l'hôpital le plus proche.

Premièrement : Il s'agit d'une urgence. Composez le 911.

Deuxièmement : En attendant l'ambulance, prenez 2 bouffées de _____ (inhalateur de soulagement) toutes les 10 minutes.

Pour chaque zone contenant des symptômes, des actions sont proposée en rapport avec ceux-ci. Le patient évalue son état suivant ces symptômes et entreprend les actions liées à celui-ci.

Les actions peuvent être des médicaments à prendre ou une personne à contacter. Celles-ci doivent pouvoir être facilement modifiable par le biais d'un panneau de configuration.

Une version complète et détaillée du présent plan d'action est disponible sur le cd joint au dossier.

L'application nécessite donc un écran de visualisation simple pour le patient incluant les symptômes et permettant un accès rapide aux actions.

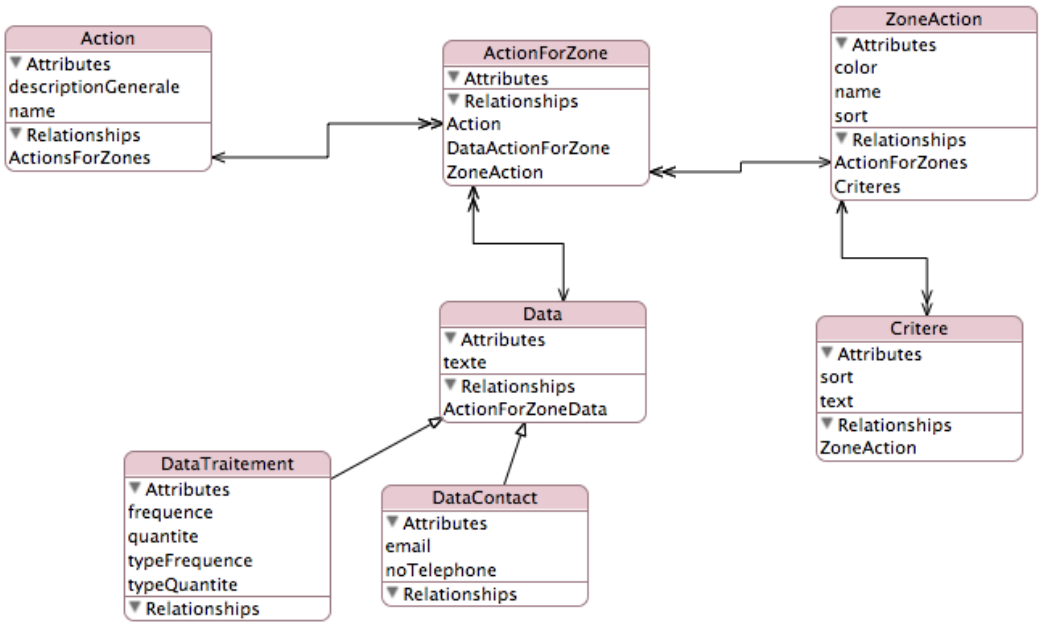
Elle nécessite également un outil de configuration pour le médecin afin que celui-ci puisse modifier les actions par rapport à son patient.

Modélisation

Ici, nous aborderons la partie de modélisation des données avec une description détaillée du modèle de données.

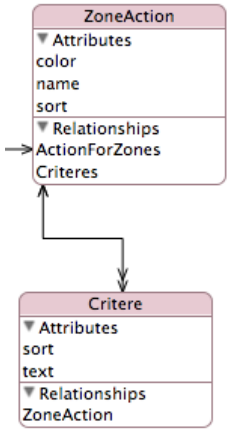
Modèle de données

Après une analyse des besoins de l'application, des données nécessitant une persistance et des extensions futures possibles, le modèle de données suivant vous est présenté, ainsi qu'une analyse détaillée de celui-ci.



Zones d'action et critères

Les zones d'action contiennent des critères. Ceux-ci représentent les symptômes qui permettent aux patients d'avoir une idée générale de leur état et ensuite les actions à entreprendre afin de gérer celui-ci.

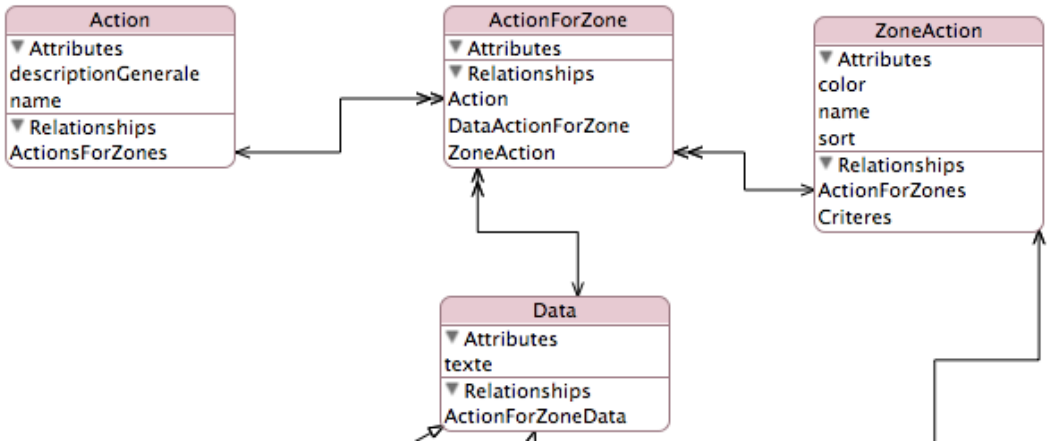


Les zones d'action comprennent un nom, une variable Sort permettant de gérer l'ordre d'apparition de celles-ci ainsi qu'une variable Color qui permettra, dans de futures améliorations, d'appliquer une couleur à l'affichage de la zone.

Les critères, quant à eux, possèdent un nom et une variable de tri.

Actions

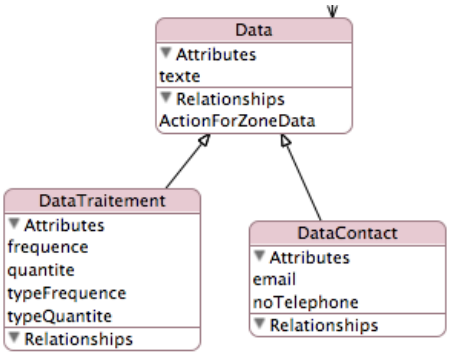
Les actions sont des traitements ou des contacts nécessaires pour le patient. Il peuvent être liés à plusieurs zones et doivent posséder des données pour chacun de ces liens.



La classe action contient un nom ainsi qu'une description générale.

La classe ActionForZone effectue le lien entre une zone et une action. Elle contient un objet de chaque type ainsi qu'un objet Data contenant les données spécifiques de l'action pour la zone donnée.

Données Zone -> Action



L'objet Data représente les données affichées pour une action dans une zone d'action. Il contient le texte à afficher pour celle-ci.

Deux types de données distinctes ont été créée a partie de celui-ci :

- **DataTraitement**
  Un traitement à prendre ponctuellement, comme un ventolin ou un médicament de contrôle.
  Ce type de données contient la fréquence de la prise ainsi que la quantité. Les champs typeFrequence et typeQuantité définissent le genre de ceux-ci.
  Par exemple : 2 comprimés 3 fois par jour
- **DataContact**
  Une personne ou un service à contacter en cas de problème ou de question, par le biais d'un téléphone ou d'un email.

Ces deux classes permettent de créer des modèles de saisie et d'affichage représentant un gain de temps pour l'utilisateur, spécialement pour le médecin lors de la configuration des actions.

Fonctionnement

Ici, nous détaillons le fonctionnement de l'application pour la partie interface patient et pour la zone de configuration.

Interface Patient

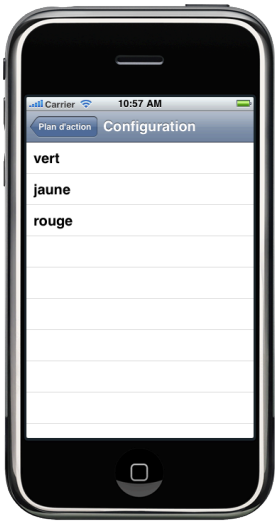
Au lancement de l'application, un écran se charge contenant les zone d'actions et les critères concernant le patient. Un système d'onglet permet à l'utilisateur de naviguer entre ces zones afin de définir dans quel état il se trouve.

En sélectionnant un des symptômes, on accède à l'écran des actions à entreprendre. Celui-ci est une liste de traitements ou de contacts visant à améliorer la situation du patient.

Interface de Configuration

Ici, nous détaillerons le fonctionnement de la zone de configuration, appelée depuis la vue principale. Elle permet au médecin de gérer les actions à entreprendre pour chaque zone.

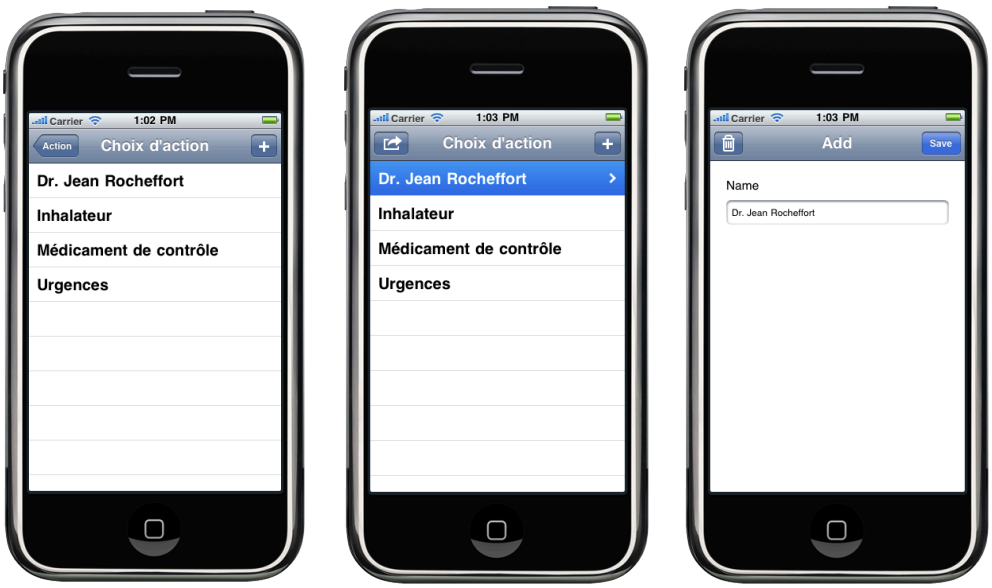
Un premier écran montre une liste des zones disponibles.



Après sélection de la zone, les actions déjà rattachées à celle-ci sont affichées dans une table view. L'utilisateur a alors la possibilité de modifier, d'ajouter ou de supprimer des actions.

Après sélection d'une action, un écran d'édition de l'action pour la zone apparaît. Il permet de gérer les données qui seront affichées pour dans un éditeur de texte.

Lorsque l'utilisateur désire ajouter une action à une zone, une vue contenant toutes les actions disponibles apparaît. Il peut ensuite choisir de la lier à la zone. Possibilité est également donnée d'ajouter une action ou de l'éditer par le biais d'un écran d'édition.



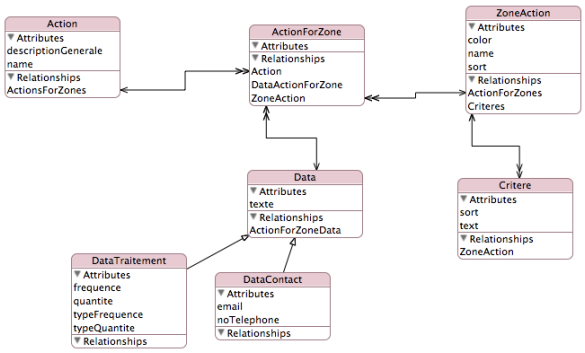
Informations techniques

L'application de plan d'action inclut les mêmes spécificités techniques que le quizz, incluant l'utilisation de TableView et de NavigationController. Dans cette partie, nous effectuerons une présentation de CoreData et de son utilisation dans l'application.

CoreData

CoreData est une aide dans la gestion des modèles d'une application Iphone ou Mac OSX. Elle fournit une gestion graphique d'un modèle de données et un outil de persistance des données.

The Managed Object Model



En utilisant le Framework CoreData, nous créons une définition abstraite de notre modèle de données. Celui-ci comprend :

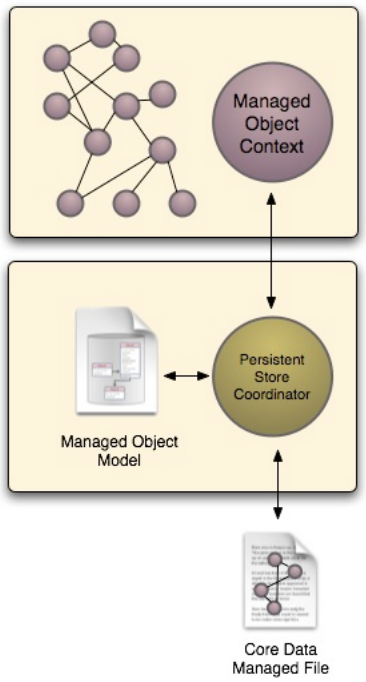
Entities
La représentation d'une table dans le modèle de données. Ex : ZoneAction

Attributes
Les composants de notre entité. Ex : Dans action, le nom

Relationships
Les relations entre les entités. Ex : une ZoneAction contient plusieurs critères

Managed Object Context and Persistent Stores

Lors de l'exécution du programme, nous accédons à notre modèle de données par le biais d'un Managed Object Context. Il gère la validation de nos objets contrôlant la correspondance avec notre Managed Object Model. Il effectue donc le lien entre notre application et notre outil de persistance des données.



Notre Managed Object Context va une utiliser un Persistant Store Coordinator. Celui, basé sur notre Managed Object Model, va gérer l'enregistrement des données sur le disque.

Ces données peuvent être enregistrées sous différentes formes :

- fichier XML
- fichier Binaire
- base de données SQLite

Application

Dans le cadre de cette application, nous avons créé le modèle de données présenté dans la partie modélisation ci-dessus.

Mise en place

Lors du chargement de l'application, nous créons notre Managed Object Context.

```
- (NSManagedObjectContext *)managedObjectContext {

    if (managedObjectContext_ != nil) {
        return managedObjectContext_;
    }

    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];
    if (coordinator != nil) {
        managedObjectContext_ = [[NSManagedObjectContext alloc] init];
        [managedObjectContext_ setPersistentStoreCoordinator:coordinator];
    }
    return managedObjectContext_;
}
```

Celui-ci contient un Persistant Store Coordinator créé à créer à partir de notre Managed Object Model.

```
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {

    if (persistentStoreCoordinator_ != nil) {
        return persistentStoreCoordinator_;
    }

    NSURL *storeURL = [NSURL fileURLWithPath: [[self applicationDocumentsDirectory]
                                               stringByAppendingPathComponent: @"IPosAction.sqlite"]];

    NSError *error = nil;
    persistentStoreCoordinator_ = [[NSPersistentStoreCoordinator alloc]
                                   initWithManagedObjectModel:[self managedObjectModel]];
    if (![persistentStoreCoordinator_ addPersistentStoreWithType:NSSQLiteStoreType
                                                   configuration:nil URL:storeURL options:nil error:&error]) {
        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }

    return persistentStoreCoordinator_;
}
```

Nous remarquons ici, que pour notre application, les données seront sauvegardées sous la forme d'une base de données SQLite.

Chargement des données

Lors de l'initialisation de la vue principale, nous allons récupérer les données enregistrées sous la forme d'un tableau de zone d'Action. Tout le modèle étant lié, nous aurons ainsi accès à tous les objets nécessaires au bon déroulement de l'application.

```
// fonction qui charge les zone Action
-(NSMutableArray *) loadZoneAction{

    NSFetchRequest *request = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription entityForName:@"ZoneAction"
                                              inManagedObjectContext:managedObjectContext];
    [request setEntity:entity];

    // Order the events by creation date, most recent first.

    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"sort" ascending:YES];
    NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:sortDescriptor, nil];
    [request setSortDescriptors:sortDescriptors];
    [sortDescriptor release];
    [sortDescriptors release];

    // Execute the fetch -- create a mutable copy of the result.

    NSError *error = nil;
    NSMutableArray *mutableFetchResults = [[managedObjectContext
                                            executeFetchRequest:request error:&error] mutableCopy];
    if (mutableFetchResults == nil) {
        // Handle the error.
        //NSLog([NSString stringWithFormat:@"%@",[error localizedDescription]]);
        return nil;
    }

    return mutableFetchResults ;

}
```

Nous effectuons donc une requête sur notre Managed Object Context demandant l'ensemble des zones d'Action. Puis, nous spécifions que celles-ci doivent être triées à partir de l'attribut sort qui gère l'ordre d'apparition de ces zones et retournons le résultat sous la forme d'un tableau.

Sauvegarde des données

Durant le déroulement de l'application, toutes les modifications s'effectueront sur le Managed Object Context. Ce n'est que lorsque l'utilisateur aura terminé d'utiliser l'application que celles-ci seront enregistrées sur le disque.

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    NSError *error = nil;
    if (managedObjectContext_ != nil) {
        if ([managedObjectContext_ hasChanges] && ![managedObjectContext_ save:&error]) {

            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
            abort();
        }
    }
}
```

L'opération de sauvegarde est lancée lorsque l'application entre en arrière plan. En effet, avec la quatrième mise à jour 4 de l'Iphone OS, la gestion du multitâche ne permet plus de quitter une application mais l'exécute en arrière plan.

Celle-ci est mise dans une file d'attente et l'OS gère lui-même la fermeture de l'application. Nous n'avons donc aucun contrôle sur l'évènement. La sauvegarde s'effectue donc lors de la mise en arrière plan de l'application.

Création des données

Lors du premier lancement de l'application, aucun enregistrement n'est présent sur le disque. Nous allons donc, durant le chargement de notre vue principale, créer les données nécessaires au fonctionnement de l'application dans le Managed Object Context.

Il s'agit des zones d'action, des critères, ainsi que des actions qui leur sont liées.

Ceci s'effectue dans la fonction ctrlData de notre RootViewController. Ci-dessous est présenté la création d'une zone d'action et d'un critère lui étant rattaché.

```
ZoneAction *z1 = (ZoneAction *)[NSEntityDescription
                                insertNewObjectForEntityForName:@"ZoneAction"
                                inManagedObjectContext:managedObjectContext];

[z1 setName:@"verte"];
[z1 setSort:[NSNumber numberWithInt:1]];
[z1 setColor:@"verte"];

NSMutableArray * newArray = [[NSMutableArray alloc]init];

Critere * temp = (Critere *)[NSEntityDescription
                             insertNewObjectForEntityForName:@"Critere"
                             inManagedObjectContext:managedObjectContext];
[temp setSort:[NSNumber numberWithInt:1]];
[temp setText:@"Je n'ai pas de toux, de respiration sifflante, d'oppression ou de souffle court."];

[newArray addObject:temp];

[z1 setCriteres:[[NSSet alloc]initWithArray:[newArray copy]]];
```

Modification et suppression des données

La modification des données s'effectue directement sur les objets chargés dans le Managed Data Context. Celui-ci, gère les objets modifiés et les propose au Persistant Store Coordinator lors de la sauvegarde.

Lors d'une suppression, nous devons la signaler au Managed Data Context. Ceci se fait simplement avec l'appel de la fonction deleteObject de celui-ci.

```
[managedObjectContext deleteObject:[myObject]];
```

Améliorations possibles

Le design des interfaces n'a pas été travaillé, il utilise la présentation standard des objets fournit par le SDK d'Apple. Une refonte de celui-ci est nécessaire afin de s'intégrer dans la ligne graphique des autres applications de ce travail de diplôme.

Les modèles de présentation des actions (Traitement et Contact) n'ont pas été utilisés. Le développement de ces parties offrirait une saisie plus rapide au médecin lors de la configuration.

Difficultés rencontrées

Deux difficultés majeures ressortent de cette partie.

Le temps à disposition était très court, 2 semaines, ce qui ne m'a pas permis de développer toutes les fonctionnalités nécessaires au programme.

Le design de l'application et la création des modèles de présentation ont été abandonnés afin de rendre une application fonctionnelle dans les délais.

La compréhension du fonctionnement du Framework CoreData et sa mise en place dans l'application a également été ardu et aurait nécessité plus de temps pour implémentation complète, notamment des parties de modèles de présentation des données.

Bilan personnel

Bilan général

La liberté accordée par ce projet a été très enrichissante. En effet, la recherche d'idée pour un cadre aussi vaste, m'a permis de laisser libre court à mon imagination. Puis, dans un deuxième temps, il a fallu recadrer les choses et d'analyser ce qui était faisable ou non dans le temps imparti.

La planification et la gestion des heures ont été particulièrement intéressantes. Dans un domaine que je connaissais en partie, le développement Iphone, devoir se donner des délais à respecter m'a permis d'avoir une meilleure idée de mes capacités. J'ai donc pu me rendre compte de la quantité de travail pouvant être effectuer dans un certain laps de temps.

Finalement, le public cible étant des enfants, construire des interfaces attractives était primordial. J'ai donc passé beaucoup de temps à les transformer et à les améliorer. Ceci me donne la sensation d'avoir atteint un degré de finition de mes applications nettement supérieur aux différents projets effectués durant ma formation.

Cependant, je suis un peu déçu de n'avoir pu terminer complètement l'application optionnelle de plan d'action. Cela me donne un léger goût d'inachevé, même si le laps de temps restant était un peu restreint.

Bilan technique

Au début de ce projet, j'avais quelques connaissances au niveau du développement Iphone. En effet, ayant déjà effectué un projet sur cette plateforme au printemps, je connaissais déjà le langage et la construction d'application classique.

Durant cet été, j'ai pu considérablement augmenter mes connaissances, notamment au niveau de l'utilisation des divers canaux d'interactivité offerts à l'utilisateur. De plus, l'utilisation de Framework CoreData m'a permis de mettre en pratique, dans un contexte concret, l'utilisation d'un outil de persistance des données.

Enfin, le fait de réaliser trois petites applications m'a permis de réajuster l'importance que je donnais aux différentes phases du développement. Au fur et à mesure de l'avancement, l'analyse, la recherche et la finition ont prisprogressivement le pas sur le développement pur.

Difficultés rencontrées

Les difficultés spécifiques aux applications ont été détaillées dans chaque partie les traitant.

D'un point de vue plus général, une grande difficulté a été le fait de se fixer des limites au projet selon le temps imparti. Atteindre un bon degré de finition des applications tout en ne perdant pas de vue les délais et la rédaction de ce rapport a demandé, de ma part, un gros effort de volonté.

Le contenu du rapport a été également source de réflexion. Il n'est pas aisé de définir ce qui est important et intéressant à présenter afin d'assurer une bonne compréhension d'un lecteur pas forcément spécialiste du domaine.

Conclusion

J'ai eu énormément de plaisir a effectué ce travail de diplôme. Le contexte et le public cible du projet global m'ont interpelé et j'espère que mes réalisations pourront faire avancer les différentes recherches dans ce domaine.

Remerciements

Je remercie mon professeur responsable, Mme Nicole Glassey, pour ses précieux conseils et pour son implication dans mon travail de diplôme.

Je remercie également l'Institut Informatique de Gestion, pour avoir mis à ma disposition un Iphone et également pour m'avoir inscrit à l'Apple Developper Program de la HESSO.

Annexes

Attestation

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré.

Cheseaux Francois :

Bibliographie et Ressources

Documentation

Documents généraux liés à la programmation IPhone

Tutoriel Iphone par M.Guillaume Fort
Accessible en annexe

Ressources Web

Documents généraux liés à la programmation IPhone

Iphone Dev Center
http://developer.apple.com/iphone/index.action

Site de tutoriaux et de développment pour Iphone
http://www.ipup.fr/

Image et graphisme de l'application

Image du docteur Tux
http://tux.crystalxp.net/fr.id.1558-tux-medecin.html

Image de ventolin
http://brownbeaver.wordpress.com/2008/10/

Image de ballon
http://www.vegea.com/images/plein-air/ballon-de-plage-publicitaire-P49002-2200.jpg

Image de Tux de profil
http://www.crystalxp.net/forum/fr/exposition-graphique/Tux-Mascottes-amp-Personnages-2/sujet_8778_1.htm

Images compte à rebours
http://a33.idata.over-blog.com/600x424/0/51/64/42/Ecole/MemoryChiffres600.jpg

Image de Fin du simulateur
http://www.merci-facteur.com/voeux/831-Bravo%20sous%20des%20ballons_maxi.gif

Sons inclus dans l'application de simulateur

Bip de réussite
http://www.universal-soundbank.com/boutons-sonores-page6.htm

Applaudissement
http://www.universal-soundbank.com/applaudissements.htm

Bruit de mélange
http://www.soungle.com/index.php?page=download&sound=7654

Core Data

Présentation du Framework CoreData
http://developer.apple.com/macosx/coredata.html

Core Data Programming Guide
http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/CoreData/cdProgrammingGuide.html

Iphone and Core Data
http://developer.apple.com/iphone/library/documentation/DataManagement/Conceptual/iPhoneCoreData01/Introduction/Introduction.html

Diverses sources utilisées

Utilisation UITouch
http://nuigroup.com/forums/viewthread/1479/

Utilisation de l'AVAudioRecorder
http://www.mobileorchard.com/tutorial-detecting-when-a-user-blows-into-the-mic/

Utilisation de l'évènement Shake
http://www.iphonedevsdk.com/forum/iphone-sdk-development/4381-iphone-shaking-code.html

Utilisation de la fonction atan2
http://pommedev.mediabox.fr/index.php?topic=4154.0

Mode d'emploi d'un ventolin
http://ww.gsk.fr/gsk/medicament/notice/Ventoline_100_susp_inhal.pdf

Exemple de plan d'action
http://www.asthma.ca/adults/control/pdf/AsthmaActionPlan.pdf

Rapport des heures

Les rapports hebdomadaires détaillés sont disponibles sur le cd proposé en annexe.

	Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5
Analyse	16	6	2		
Recherche et test		6			
Développement			14	20	10
Design					10
Rédaction					
Total	16	12	16	20	20

	Semaine 6	Semaine 7	Semaine 8	Semaine 9	Semaine 10
Analyse	8				8
Recherche et test		28			16
Développement		6	22		
Design			8	6	
Rédaction	24			26	6
Total	32	34	30	32	30

	Semaine 11	Semaine 12	Semaine 13	Total
Analyse				**46**
Recherche et test	8	10		**62**
Développement	14	27	20	**133**
Design				**24**
Rédaction			36	**92**
Total	22	37	56	***357***

Projet général

Ipos Quizz

Ipos Simulator

Ipos Action

Cahier des charges

Projet : IPos

Hes-So Valais / Travail de Bachelor 2010

François Cheseaux

Introduction

L'objectif de ce travail de diplôme est d'explorer les possibilités offertes par une application Iphone devant permettre à un enfant de comprendre sa maladie et d'apprendre sa posologie de manière ludique.

Sommaire

Table des matières

Description générale

Ce projet s'inspire librement du fonctionnement de l'école de l'asthme de Sion, le public cible sera donc des enfants.

Le jeu a été choisi comme canal de communication afin de permettre un apprentissage ludique de sa maladie et de la manière de la gérer.

Dans cet optique, deux idées seront développées :

- un quizz d'apprentissage permettant de connaître sa maladie et sa posologie
- un simulateur de ventolin utilisant les possibilités ergonomiques de l'Iphone

En suivant la logique d'une école, une interface de gestion des questions et des résultats du quizz sera mis à disposition de l'enseignant.

Quizz d'apprentissage

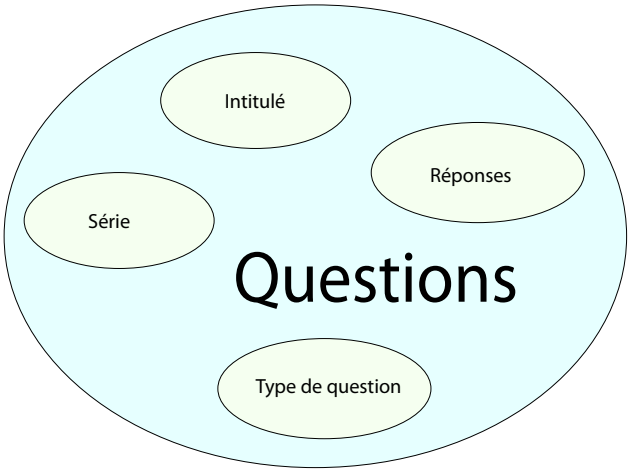
Fonctionnalités

Afin d'offrir un apprentissage varié, le quizz proposera plusieurs formes de questions ainsi qu'un écran de visualisation des résultats.

Les questions seront réparties en plusieurs séries pouvant correspondre à un thème ou à un niveau de difficulté.

L'interface graphique de l'application devra être attractive afin de correspondre à notre public cible.

Structure d'une question



Une question sera composée de quatre éléments :

- Un intitulé soit la question elle-même
- Un type de question *(voir plus bas)*
- Une ou plusieurs réponses selon le type de question choisi
- Une série, soit un thème ou un niveau de difficulté

Type de questions

Afin d'offrir un apprentissage ludique, trois types de questions ont été retenues :

Question duo

Une question offrant deux possibilités de réponse.

Question quatuor

Une question offrant quatre possibilités de réponse.

Question Scrabble

Les lettres composant la réponse sont disséminées sur l'écran et l'utilisateur doit les remettre en place.

Fonctionnalités

En utilisant les possibilités ergonomiques de l'Iphone, nous allons créer un simulateur de ventolin permettant l'apprentissage de son utilisation.

Le scénario sera basé sur le mode d'emploi de l'appareil afin de correspondre le plus précisément à la réalité.

Scénario

Le scénario se découpera en 5 étapes :

- Positionner le ventolin
  En tournant l'Iphone mettre le ventolin dans la bonne position

- Secouer le ventolin afin de mélanger le liquide
  En secouant l'Iphone simuler le mélange du liquide

- Se vider les poumons
  En soufflant dans le micro de l'Iphone se vider les poumons

- Mettre le ventolin dans la bouche et inspirer
  Viser la bouche du bonhomme apparaissant à l'écran et appuyer sur l'extrémité du ventolin
- Retenir sa respiration durant 10 secondes
  Grâce à un compte à rebours à l'écran retenir sa respiration durant dix secondes

Un scénario graphique est mis en annexe du présent document.

Partie Optionnelle

Interface de gestion

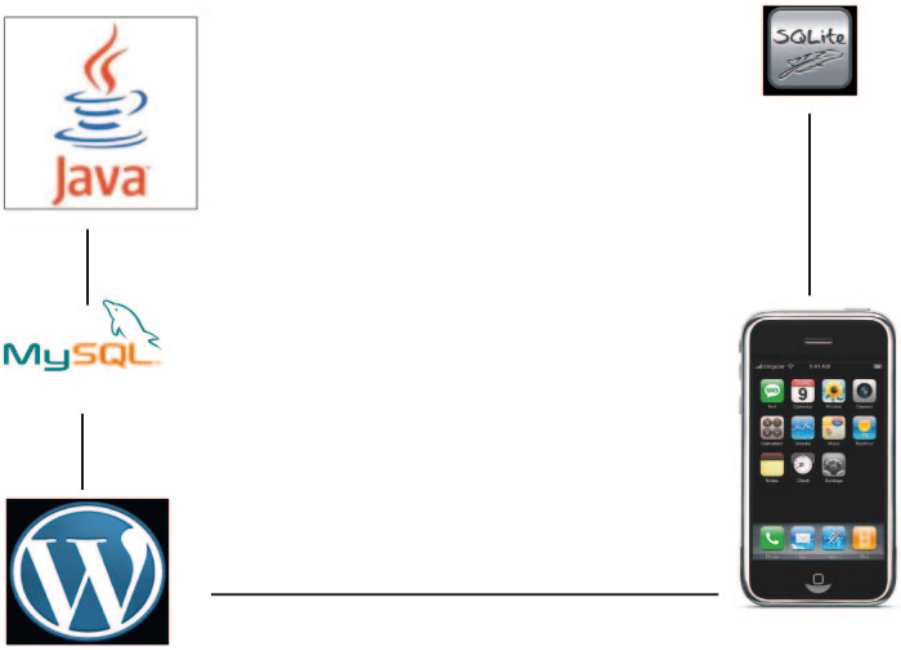
Fonctionnalités

L'objectif est de fournir à l'enseignement une interface lui permettant de créer des questions et séries de questions et de les distribuer sur les Iphone des élèves.

Elle devra également lui permettre de réceptionner les résultats des élèves et lui afficher des statistiques de ceux-ci.

Architecture

Le transfert de donnée se fera par le biais d'un web service installé sur le poste de l'enseignant. L'interface du professeur sera une application java. Les données seront stockées dans une base de donnée mySql.



Planning

Le détail de la planification est disponible en annexe.

Signature

Par leur signature, les soussignés déclarent avoir lu et approuvé l'entier du document.

Nicole Glassey Date : …… Signature : ……

François Cheseaux Date : …… Signature : ……

Sierre, le 16 août 2010