



École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

**Département Informatique
4ieme année
2011 - 2012**

Rapport projet collectif

**Application d'aide à l'identification
d'insectes nuisibles**

Encadrants

Gilles VENTURINI
gilles.venturini@univ-tours.fr

Université François-Rabelais, Tours

Client

Ingrid ARNAULT
ingrid.arnault@univ-tours.fr
Damien MUNIER
munier.damien@aliceadsl.fr
Alexandre DEPOILLY
alexandre.depoilly@etu.univ-tours.fr

Équipe CETU INNOPHYT

Étudiants

Matthieu ANCERET
matthieu.anceret@etu.univ-tours.fr
Jérôme HEISSLER
jerome.heissler@etu.univ-tours.fr
Julien TERUEL
julien.teruel@etu.univ-tours.fr
Martin DEMEULEMEESTER
martin.demeulemeester@etu.univ-tours.fr
Mickael PURET
mickael.puret@etu.univ-tours.fr
Simon FAUSSIER
simon.faussier@etu.univ-tours.fr
Zheng ZHANG
zheng.zhang@etu.univ-tours.fr
Zhengyi LIU
zhengyi.liu@etu.univ-tours.fr

DI4 2011 - 2012

Version du June 4, 2012

Contents

1	Introduction	6
2	Présentation du projet	7
3	Analyses complémentaires	8
3.1	Modélisation UML	8
3.2	Le fichier XML et sa DTD	8
4	Application PC	14
4.1	Interface et utilisation	14
4.1.1	Bandeau des questions	15
4.1.2	Bandeau des médias d'une question	15
4.1.3	Bandeau des réponses	15
4.1.4	Bandeau d'affichage des résultats et des médias	15
4.2	Module de lecture/écriture XML	16
4.2.1	Déroulement d'une lecture XML	17
4.2.2	Déroulement d'un enregistrement XML	17
4.3	Mécanisme d'affichage des arborescences	17
4.4	Visualisation des médias	19
4.4.1	Visualisation des images	19
4.4.2	Lecteur audio	19
4.4.3	Lecteur vidéo	21
4.5	Algorithme de coloration des questions	22
5	Application mobile	23
5.1	Fonctionnalités	23
5.2	La zone d'affichage	24
5.3	La caméra	25
5.4	Les bitmaps	27
5.5	L'historique	28
5.6	La gestion des projets	29
5.6.1	D'un point de vue utilisateur	29
5.6.2	Structure du logiciel	30
6	Aspect gestion de projet	36
6.1	Témoignage de l'équipe	36
6.2	Calendrier prévisionnel et calendrier réel	39
7	Futur du projet	42
8	Conclusion	44

A Documentation utilisateur de l'application mobile	45
A.1 Introduction	45
A.2 Objectif de l'application	45
A.3 Créer et configurer un compte	45
A.4 L'identification	48
A.5 L'identification rapide	50
A.6 L'exportation	51

List of Figures

3.1	Diagramme de classes de l'application PC	11
3.2	Version finale du fichier XML	12
3.3	DTD de notre fichier XML	13
4.1	Interface principale de l'application au démarrage	14
4.2	Interface principale après avoir sélectionné une question	16
4.3	Boite de dialogue lorsque l'on veut créer/modifier une question	16
4.4	Code permettant d'afficher une image	19
4.5	Capture d'écran du lecteur audio	20
4.6	Capture d'écran du lecteur vidéo	21
4.7	Coloration des questions selon la quantité d'informations présentes	22
5.1	Menu d'accueil de l'application mobile	24
5.2	Interface d'identification d'un insecte - application mobile	25
5.3	Schéma explicatif du fonctionnement des fragments	26
5.4	Code permettant d'éviter les problèmes liés à la caméra	27
5.5	Mise à jour de l'objet caméra et rafraîchissement de la zone de visualisation de la classe CameraPreview	28
5.6	Destructeur de la classe ImageAdapter	28
5.7	Code pour passer à la question suivante	29
5.8	Structure arborescente de la gestion des utilisateurs	30
5.9	Interface d'identification d'un utilisateur	31
5.10	Interface de gestion des campagnes	32
5.11	Interface de gestion des parcelles	33
5.12	Interface de gestion des pièges	33
5.13	Modélisation de la base de données	34
5.14	Modélisation des interactions entre la BDD et l'application	35
6.1	Calendrier prévisionnel par tâches	40
6.2	Calendrier réel par tâches	41
A.1	Interface de connexion à l'application mobile	46
A.2	Interface de sélection d'une campagne	47
A.3	Interface d'identification d'un insecte	49
A.4	Interface d'affichage du résultat	50
A.5	Mosaïque des insectes pour l'identification rapide	51
A.6	Sauvegarde du résultat de l'identification rapide	52
A.7	Exportation des résultats au format CSV	52

Introduction

Notre projet est un projet collectif à 8 personnes dans le cadre de notre cursus à Polytech'Tours. Le client de notre projet est l'équipe INNOPHYT, et plus particulièrement la responsable de cette équipe, Ingrid Arnault. Cette équipe fait partie de l'Université François Rabelais de Tours et elle est consacrée aux activités de valorisation et de recherche dans le domaine de la lutte antiparasitaire durable. Gilles Venturini, professeur au sein de l'école Polytech'Tours, est notre encadrant de projet. De par sa bonne connaissance du projet, il est aussi un interlocuteur privilégié pour les aspects techniques et opérationnels. Ce rapport est un guide technique, qui pourra servir à un développeur désireux de reprendre et de comprendre les applications existantes, mais aussi un guide utilisateur, qui permettra à l'usager de trouver les réponses à ses questions à propos du fonctionnement des logiciels.

Présentation du projet

L'objectif du projet est de réaliser deux applications :

- Une application sur PC qui permet de consulter la base de données (l'arbre de décision) et de la faire évoluer (ajout/modification/suppression d'espèces, de questions, de réponses, de médias...). Cette application sera principalement utilisée par l'équipe INNOPHYT ou éventuellement par des experts du domaine. Elle permet de visualiser rapidement les éléments sur lesquels il manque des informations. Elle permet aussi d'exporter la base de données pour la mettre à disposition de l'application mobile.
- Une application mobile qui permet à l'utilisateur, directement sur le terrain, de déterminer le type de l'insecte et de décider s'il est nuisible ou non. L'application va poser un certain nombre de questions et va déduire des choses à partir des réponses obtenues (via l'arbre de décision). Celle-ci propose à l'utilisateur différents médias (photos, vidéos, sons, animations...) pour l'aider à prendre une décision. Elle permet aussi de fournir un rapport des actions effectuées sur celle-ci (espèces inconnues, questions/réponses floues, résultat...) pour permettre à l'équipe INNOPHYT de corriger ou de faire évoluer la base.

Nous devons donc produire un logiciel d'aide à la décision le plus pertinent possible pour l'utilisateur. Actuellement, l'équipe INNOPHYT utilise un fichier tableur (Excel), construit sous forme arborescente et contenant les questions et les réponses. Ce type de fichier est assez difficile à maintenir et à faire évoluer facilement. De plus, ce n'est pas un format adapté pour la visualisation et l'exploitation de ce type de données (arborescence). Les futurs utilisateurs pouvant être aussi bien des scientifiques du domaine que des utilisateurs lambda, il est primordial de proposer un outil accessible et utilisable pour tout le monde (familiarisé à l'informatique ou non), et ce, particulièrement pour l'application mobile.

Analyses complémentaires

Nous allons présenter ici les modélisations et diagrammes UML réalisés a posteriori du cahier de spécifications. En effet, lors de la rédaction du cahier de spécification, nous n'avions pas encore modélisé la partie "intelligence" des applications.

3.1 Modélisation UML

Avant de développer l'application PC, nous avons pris soin de modéliser une structure fiable et efficace à même de réaliser toutes les fonctionnalités demandées et de pouvoir évoluer de façon simple. La figure 3.1 explique l'architecture de notre application PC.

Ce diagramme de classe illustre l'ensemble des objets de notre application et la manière dont ils communiquent entre eux. Étant donné que nous travaillons sur une arborescence, il a fallu trouver un modèle pour représenter cet arbre. Le fichier XML représentant déjà un arbre avec la notion de noeud parent et enfant, nous avons associé chaque balise XML à une classe. Petit rappel sur les équivalents entre diagramme de classe et fichier XML :

Nommage fichier XML	Nommage modélisation UML
branche	Catégorie
question	Question
legende	Media
img	Media
audio	Media
video	Media
reponse	Reponse
resultat	Especie

On remarque que la balise "media" n'a pas de classe associée. En effet, cette balise servant simplement à délimiter l'ensemble des médias (legende, img, audio...) d'une question, d'une réponse ou d'un résultat, nous n'avons pas à nous en occuper.

Maintenant que nous avons identifié les classes que nous allons utiliser, il faut les remplir avec des attributs. Outre les informations propres à chacune des classes, par exemple le texte associé à une question, il faut penser à garder la notion d'arborescence (c'est-à-dire la notion de parenté). Pour cela, chaque classe possède un attribut qui pointe vers son père et une liste contenant des pointeurs vers ses fils. C'est ici qu'interviennent nos classes ListeXXXX. Elles sont utile pour représenter la notion de parenté, mais elles nous permettent aussi de ne pas avoir à re-lire notre fichier XML à chaque recherche des fils d'un objet. Le seul point négatif de cette méthode est qu'elle nous oblige à charger en mémoire la totalité de l'arbre au lancement de l'application. Mais cela nous permet aussi, en cas de modification d'éléments, de différer la sauvegarde (tout est en mémoire). De ce fait, il y a plus de souplesse dans la gestion de l'enregistrement et dans la lecture du fichier XML.

3.2 Le fichier XML et sa DTD

Notre base de données est représentée sous la forme d'un fichier XML. Ce format a été privilégié car il est relativement simple à manipuler et assez performant pour de petites bases, comparativement à une base

SQL/SQLite par exemple. N'ayant de toute manière pas besoin de relation, d'intégrité et d'indexation, la choix du XML s'est imposé de lui-même.

En raison de nouveaux éléments apparus tout au long du projet (après la rédaction du cahier de spécification), la structure du fichier XML de données a évoluée. Par exemple, nous avons inséré un nouvel attribut "visible" sur les questions et les réponses suite à la demande du client de voir apparaître ce critère dans l'interface (dans le but d'aider l'utilisateur de l'application). De plus, au fur et à mesure de l'avancement du projet, nous nous sommes rendu compte de quelques erreurs ou incohérence dans le fichier XML. Nous avons donc du corriger ces problèmes, ce qui a engendré des modifications au niveau du code C++ de traitement du fichier XML.

Le fichier XML final est représenté sur la figure 3.2.

Nous avons bien sur corrigé la DTD associée à ce fichier XML (cf. figure 3.3). Cette DTD est le modèle, la grammaire, du fichier XML et permet donc de le valider. Elle indique les propriétés de chaque balise ainsi que l'arborescence de notre fichier.

Nous allons passer en revue les différentes balises utilisées dans cette DTD :

Balise <!ELEMENT>

La balise de type ELEMENT nous indique que nous aurons une nouvelle balise dans notre fichier XML. C'est ce que l'on appelle un élément.

- (**question**) : cet opérateur nous indique que l'élément contiendra forcément un unique élément question
- (**reponse+**) : cet opérateur nous indique que l'élément contiendra au moins un élément réponse
- (**branche|resultat**) : cet opérateur nous indique que l'élément contiendra un élément branche ou d'un élément résultat
- (**nom, type**) : cet opérateur nous indique que l'élément contiendra un élément nom et d'un élément type
- (**media***) : cet opérateur nous indique que l'élément contiendra 0 ou plusieurs éléments media
- (**#PCDATA**) : cet opérateur indique que l'élément contiendra une donnée, généralement un texte en relation avec le nom utilisé par l'élément
- **ANY** : cet opérateur indique que tout type d'élément est autorisé

Balise <!ATTLIST>

La balise ATTLIST nous indique que l'élément qui lui est relié aura des attributs. Les attributs sont des données indiquées à l'intérieur des balises dans le fichier XML. Elles donnent généralement des détails sur l'élément qui ne seront pas considérés comme des données.

- **#IMPLIED** : indique que l'attribut est facultatif pour l'élément
- **#REQUIRED** : indique que l'attribut est obligatoire pour l'élément
- **CDATA** : indique que l'attribut sera une chaîne de caractères
- **NMTOKENS** : indique que l'attribut sera une chaîne de caractères réduite (seule une partie des caractères sont autorisés)
- **ENTITY** : indique que l'attribut sera une entité externe qui ne sera pas analysée

3.3 Exportation du résultat de l'application mobile

Pour que l'utilisateur puisse donner le résultat de son travail à l'équipe INNOPHYT, nous avons mis en place un système d'exportation de la base de données vers un fichier au format CSV (avec le point-virgule comme signe de séparation des colonnes). Cette méthode n'exporte que les données de l'utilisateur qui en fait la demande. Le résultat est enregistré dans le dossier innophyt/export. Les informations exportées sont les suivantes :

- L'id et le nom de l'utilisateur

- L'id, le nom, la date de début et de fin, l'adresse, les coordonnées (latitude/longitude) et la description de la campagne
- Même chose pour les parcelles et les pièges
- L'id, le nom et le nombre d'insectes

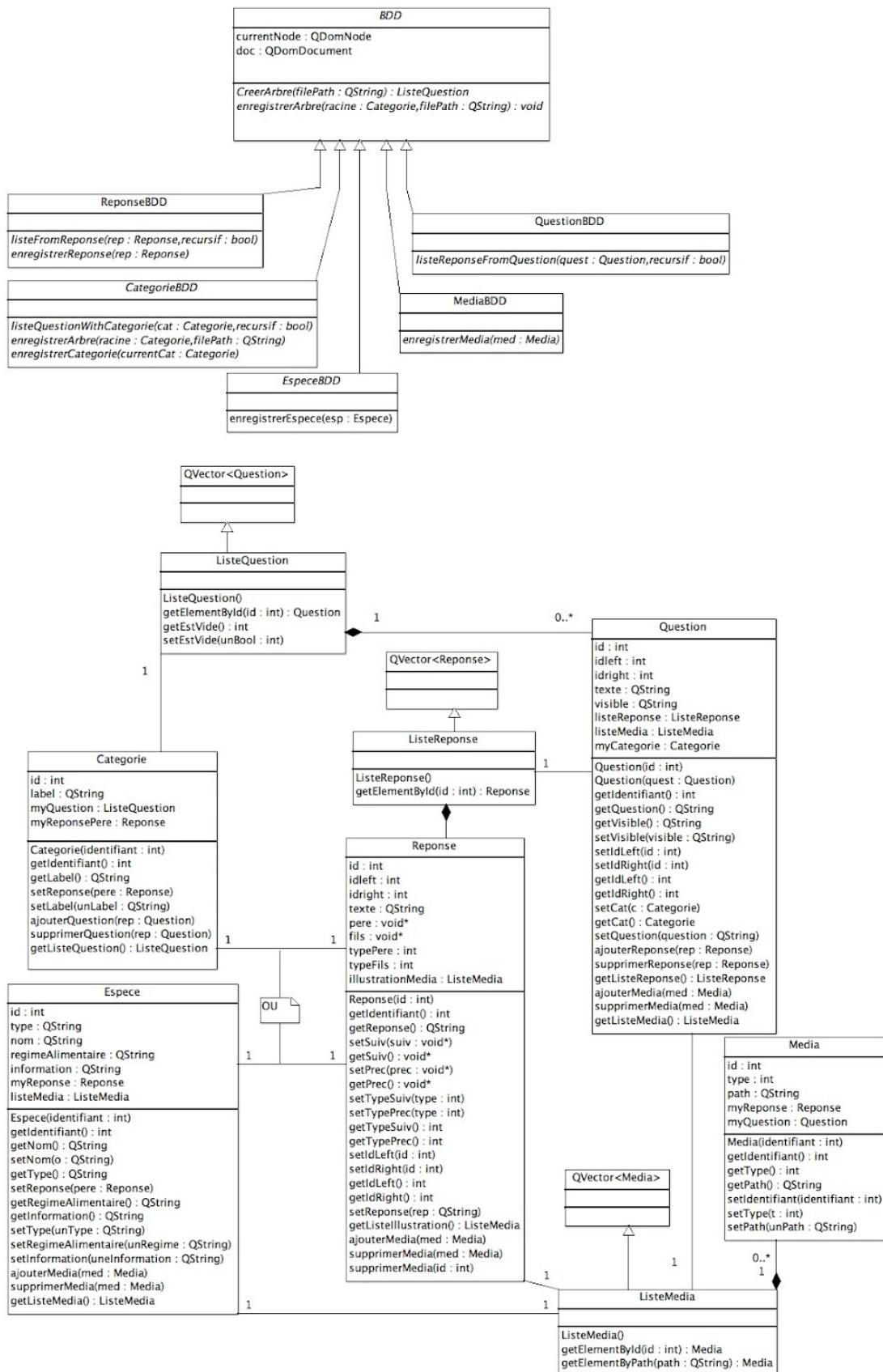


Figure 3.1: Diagramme de classes de l'application PC

```

<?xml version = '1.0' encoding="UTF-8"?>
<!DOCTYPE arbre SYSTEM "arbre.dtd">

<arbre>
  <branche id="b1" type="accueil" date="01/06/2012">
    <question id="q1" texte="Premiere question ?" visible="true">
      <media>
        <legende>Astuces : regarder attentivement l'insecte</legende>
        
        <audio id="audi1" src="images/test_audio.wav" />
      </media>
      <reponse id="r1" texte="Reponse 1">
        <media>
          
          
          <legende>Ceci est le texte de la reponse 1</legende>
        </media>
      <branche id="b2">
        <question id="q3" texte="Autre question" visible="false">
          ...
        </question>
      </branche>
    </reponse>
    <reponse id="r2" texte="Reponse 2">
      <media>
        
        <legende>Ceci est le texte de la reponse 2</legende>
      </media>
      <branche id="b3" type="Arachnide">
        <question id="q2" texte="Deuxieme question ?" visible="true">
          <reponse id="r3" texte="Reponse 3">
            <resultat id="res1">
              <nom>Insecte 1</nom>
              <type>MEL1</type>
              <regimeAlimentaire>Predateur</regimeAlimentaire>
              <informations>Informations insecte 1</informations>
              <media>
                
              </media>
            </resultat>
          </reponse>
          <reponse id="r4" texte="Reponse 4">
            <branche id="b2">
              <question id="q6" texte="Encore une question ?" visible="both">
                ...
              </question>
            </branche>
          </reponse>
        </question>
      </branche>
    </reponse>
  </branche>
</arbre>

```

Figure 3.2: Version finale du fichier XML

```
<!ATTLIST branche date CDATA #IMPLIED>

<!ELEMENT branche (question)>
<!ATTLIST branche id ID #REQUIRED>
<!ATTLIST branche type CDATA #IMPLIED>

<!ELEMENT question (reponse+, media*)>
<!ATTLIST question id ID #REQUIRED>
<!ATTLIST question texte NMTOKENS #REQUIRED>
<!ATTLIST question visible NMTOKENS #IMPLIED>

<!ELEMENT reponse ((branche|resultat), media*)>
<!ATTLIST reponse id ID #REQUIRED>
<!ATTLIST reponse texte NMTOKENS #REQUIRED>
<!ATTLIST reponse visible NMTOKENS #IMPLIED>

<!ELEMENT resultat (nom, type, regimeAlimentaire, informations, media*)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT regimeAlimentaire (#PCDATA)>
<!ELEMENT informations (#PCDATA)>

<!ELEMENT media (img+, video+, audio+, legende+)>
<!ELEMENT img EMPTY>
<!ATTLIST img id ID #REQUIRED>
<!ATTLIST img src ENTITY #REQUIRED>
<!ELEMENT video EMPTY>
<!ATTLIST video id ID #REQUIRED>
<!ATTLIST video src ENTITY #REQUIRED>
<!ELEMENT audio EMPTY>
<!ATTLIST audio id ID #REQUIRED>
<!ATTLIST audio src ENTITY #REQUIRED>
<!ELEMENT legende (#PCDATA)>
```

Figure 3.3: DTD de notre fichier XML

Application PC

4.1 Interface et utilisation

Nous devions afficher une grande quantité de données à l'écran (questions, réponses, résultats et médias), sous une forme pas forcément évidente (arborescence). Nous avons donc conçu une interface sous forme de 4 bandeaux verticaux contenant chacun un type d'information. Cette disposition permet d'avoir accès à toutes les informations nécessaires en même temps à l'écran.

La figure 4.1 permet de voir l'interface principale de l'application au démarrage (les questions ne sont pas déroulées). On remarque les 4 bandeaux, qui, de gauche à droite, servent à : gérer les questions, gérer les médias associés à la question courante, gérer les réponses et les médias des réponses de la question courante et enfin d'afficher les images et les résultats.

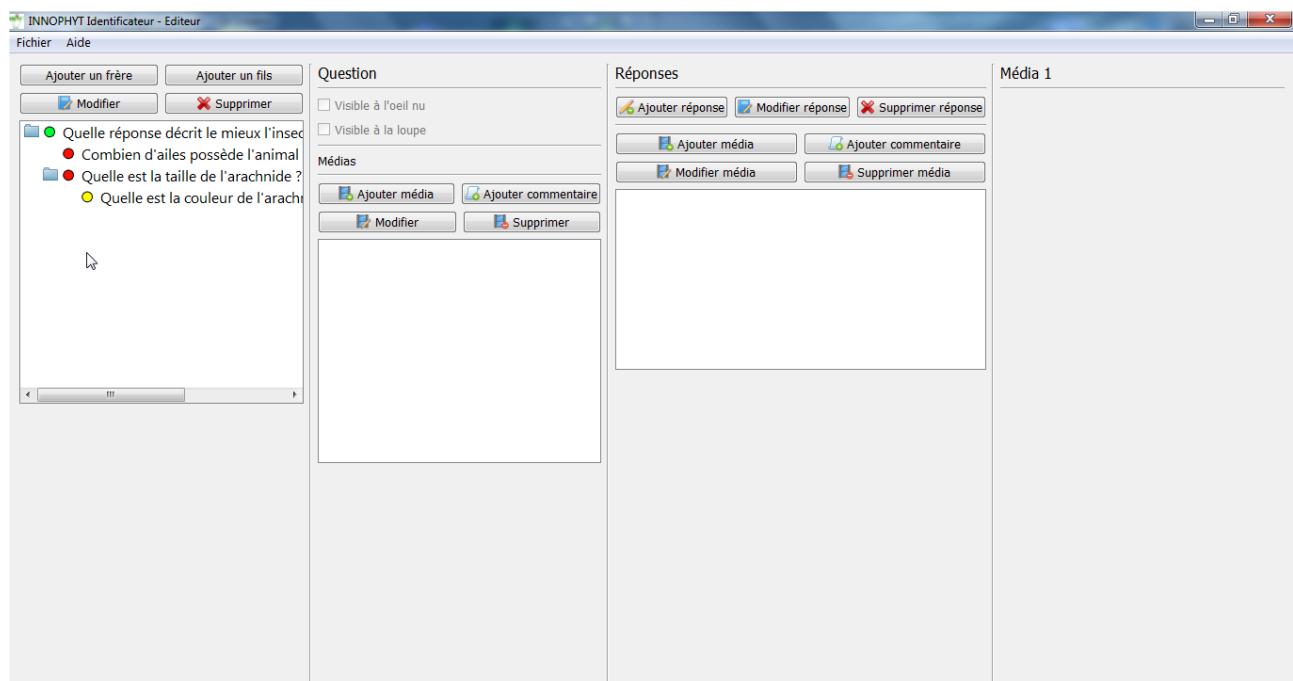


Figure 4.1: Interface principale de l'application au démarrage

Sur la figure 4.2, on peut voir que les autres bandeaux sont remplis. En effet, l'utilisateur ayant sélectionné une question, les médias et les réponses associés à cette question vont venir remplir les bandeaux correspondants. De plus, on voit qu'une image apparaît dans le bandeau le plus à droite. C'est le résultat du clic sur le média "insecte.jpg" dans le bandeau des médias de la question.

Nous allons maintenant passer à une explication, bandeau par bandeau, de chaque bouton de l'interface.

4.1.1 Bandeau des questions

- **Ajouter un frère** : permet d'ajouter une nouvelle question au même niveau (mais en dernière position) que la question actuellement sélectionnée. Si l'on se réfère à l'arbre XML, cette question est le frère de la question sélectionnée. Lors du clic, une fenêtre va s'ouvrir permettant de rentrer le texte de la question, ainsi que les critères "visible à l'oeil nu" et "visible à la loupe".
- **Ajouter un fils** : permet d'ajouter une nouvelle question en tant que fils de la question sélectionnée. Le fonctionnement est identique à "Ajouter un frère".
- **Modifier** : permet d'afficher une fenêtre, avec ses champs pré-remplis par rapport à la question sélectionnée.
- **Supprimer** : permet de supprimer une question de la liste.

4.1.2 Bandeau des médias d'une question

- **Ajouter média** : cette fonction ouvre une fenêtre de sélection de fichier qui permet de choisir un média à ajouter à la question sélectionnée.
- **Ajouter commentaire** : cette fonction affiche une fenêtre permettant de taper le texte que l'on souhaite ajouter en tant que média de la question sélectionnée.
- **Modifier** : permet de modifier le média sélectionnée.
- **Supprimer** : permet de supprimer le média sélectionné.

4.1.3 Bandeau des réponses

- **Ajouter réponse** : Permet d'ajouter une nouvelle réponse à la question sélectionnée. Cette réponse est ajoutée à la fin de la liste des réponses.
- **Modifier réponse** : Permet de modifier le contenu d'une réponse.
- **Supprimer réponse** : Permet de supprimer une réponse.
- **Ajouter média** : Permet d'ajouter un média à la réponse sélectionnée. Ce média est ajouté à la fin de la liste des médias.
- **Ajouter commentaire** : Permet d'ajouter un média textuel à la réponse sélectionnée.
- **Modifier média** : Permet de modifier le média sélectionnée.
- **Supprimer média** : Permet de supprimer le média sélectionné.

Attention, lorsque l'on clic sur un bouton "Supprimer", aucun confirmation n'est demandée.

4.1.4 Bandeau d'affichage des résultats et des médias

Ce bandeau permet d'afficher les médias images lorsqu'ils sont sélectionnés dans un des autres bandeaux. Il permet aussi d'afficher les résultats, si lorsque l'on clic sur une réponse, celle-ci débouche sur un résultat.

Lorsque l'utilisateur veut créer/modifier une question, il aura affaire à la fenêtre présente sur la figure 4.3. Cette fenêtre lui permet d'écrire le texte de la question et de choisir si cette question est visible à l'oeil nu, à la loupe ou les deux via les checkbox. En validant la fenêtre, il crée/modifie la question sélectionnée.

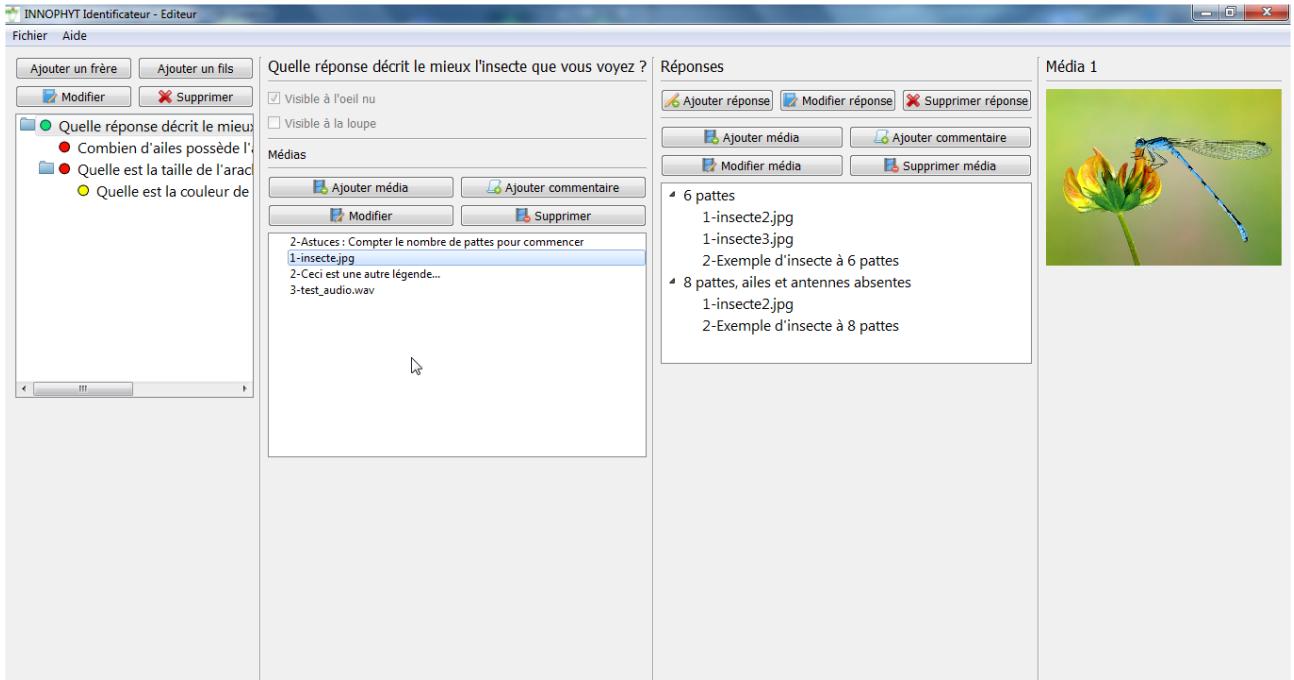


Figure 4.2: Interface principale après avoir sélectionné une question

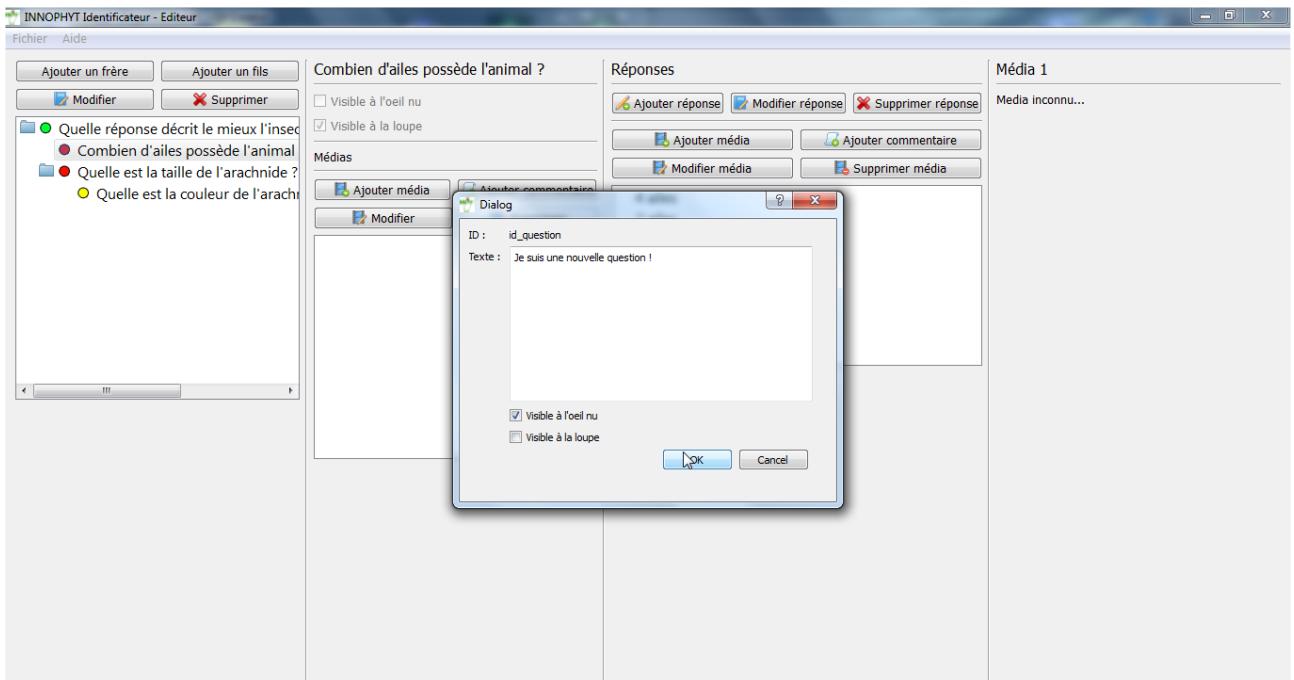


Figure 4.3: Boîte de dialogue lorsque l'on veut créer/modifier une question

4.2 Module de lecture/écriture XML

Dans cette partie, nous allons nous intéresser à la lecture d'un fichier XML. Étant donné que notre diagramme UML est circulaire, nous allons avoir à traiter plusieurs fois les mêmes balises à des profondeurs différentes dans le fichier XML. Pour éviter la redondance du code, nous avons découpé en 4 sous-classes la classe gérant le fichier XML. Chaque classe gère une autre classe bien spécifique : par exemple Ques-

tionBDD s'occupe de lire et écrire un objet Question avec des fils. Nous avons ensuite créé pour chaque classe des méthodes réalisant une lecture ou une écriture. Cependant, pour pouvoir parcourir l'arborescence d'un fichier XML, nous avons besoin de connaître le noeud à partir duquel nous lisons/écrivons des fils. C'est pourquoi nous avons ajouté un attribut "static currentNode" à la classe BDD, qui est utilisé par toutes les classes filles. Grâce à cet attribut, on peut écrire un code générique qui effectue des opérations sur le fichier XML.

4.2.1 Déroulement d'une lecture XML

Au départ, nous partons du noeud racine du fichier XML, c'est-à-dire la balise "arbre". Ensuite, nous allons avoir différentes branches en fonction de la catégorie de l'insecte. Nous allons donc utiliser CatégorieBDD pour lister nos branches. Afin de ne pas perdre le fil, nous mettons à jour notre variable currentNode. Ensuite, dans chaque branche, nous pouvons avoir une liste de questions. Pour récupérer ces questions, nous allons utiliser QuestionBDD qui va lire toutes les questions depuis le noeud père pointé par currentNode. Puis, pour chaque question, nous allons modifier la valeur de currentNode et lire la liste de réponse associée à chaque question en utilisant notre classe ReponseBDD. Enfin, il ne faut pas oublier qu'une réponse ou une question peut avoir des médias, d'où l'appel à la classe MediaBDD avant de parcourir les fils. Pour finir, suite à une question, on peut arriver sur un résultat (une espèce) et dans ce cas nous utiliserons EspeceBDD, ou sur une catégorie d'espèce et dans ce cas on recommence la lecture en rappelant CatégorieBDD.

Nous faisons cette démarche à chaque fois, de façon récursive, pour traiter nos éléments de façon logique. De cette façon, la partie lecture sera opérationnelle. Nous allons maintenant voir la partie enregistrement.

Afin de pouvoir ajouter des questions, des réponses et médias, il a fallu gérer l'enregistrement de notre arbre dans un fichier XML. Pour ce faire, nous allons utiliser la même structure que pour la lecture. Cependant, nous allons recréer des fonctions EnregistrerXXX en fonction de ce qu'il faut enregistrer. Ces fonctions seront créées dans les classes CatégorieBDD, ReponseBDD, QuestionBDD, MediaBDD et EspeceBDD. Ainsi, nous allons construire un fichier XML qui aura la même structure que notre fichier XML d'origine.

4.2.2 Déroulement d'un enregistrement XML

Dans un premier temps, nous créons une balise "arbre" grâce à la fonction BDD::enregistrerArbre(). Ensuite, nous créons ses fils, les balises "branches", avec la fonction CatégorieBDD::enregistrerCatégorie(). Nous avançons en profondeur d'abord de la même façon pour la lecture et nous écrivons les balises "questions" grâce à QuestionBDD::enregistrerQuestion(). Nous gardons la même procédure pour les balises "réponses". Évidemment, il ne faut pas oublier de mettre à jour au fur et à mesure notre variable currentNode. Puis, selon le type du fils, nous enregistrons une espèce (un résultat) via EspeceBDD::EnregistrerEspece() ou une catégorie avec CatégorieBDD::EnregistrerCatégorie(). Sans oublier que pour les questions, les réponses et les espèces, il faut enregistrer les médias qui leur sont associés grâce à MediaBDD::EnregistrerMedia.

4.3 Mécanisme d'affichage des arborescences

Pour visualiser les questions, les réponses et les médias, nous avons utilisé le composant Qt QTreeView. Ce composant est particulièrement adapté pour visualiser des structures arborescentes. Il fonctionne sur le principe du modèle Vue/Contrôleur. Nous allons aborder rapidement le fonctionnement de ce composant :

- **Création du QTreeView :** ceci est réalisé via le créateur d'interface de QtCreator.

- **Création du modèle :**

```
QStandardItemModel model = new QStandardItemModel;
```

```
ui->unTreeView->setModel(model);
```

- **Récupération de l'élément sélectionné :** il faut implémenter un mécanisme de Signal/Slot sur le composant QTreeView et intercepter le signal "clicked". Ce signal fournit comme paramètre un QModelIndex. Celui-ci permet de récupérer la ligne et la colonne de l'élément. On peut aussi, à partir de cet index, récupérer l'élément en lui-même grâce à la fonction "itemFromIndex()" du modèle du treeview.
- **Ajout d'un élément :**

```
QStandardItem elem = new QStandardItem(icon, texte); // le paramètre "icon" est optionnel
model->appendRow(elem);
```
- **Modifier un élément :** il faut récupérer l'élément (QStandardItem), le modifier via les setteurs correspondants et le modifier dans le modèle du treeview.
- **Supprimer un élément :** le modèle propose une méthode "remove()" prenant en paramètre la ligne et l'élément parent de l'élément à supprimer.

La vue est automatiquement mise à jour lorsque le modèle est changé.

Au démarrage de l'application, nous chargeons le fichier XML en mémoire et nous remplissons le treeview des questions (le bandeau le plus à gauche). Les autres treeview sont remplis en fonction de la question cliquée par l'utilisateur.

Étant donné que nous ne stockons que le texte de la question dans les éléments du treeview, nous devions trouver une solution pour pouvoir retrouver l'objet "Question" lors du clic dans le treeview. Pour cela, nous avons décidé de stocker les objets questions dans une QMap, avec comme clé les coordonnées sous forme de QString. Cela nous permet, à partir des coordonnées de l'élément sélectionné que nous pouvons calculer facilement, de retrouver la question associée. Grâce à celle-ci, nous pourrons remplir les autres treeview. Pour les coordonnées, nous sommes partis sur le principe suivant (les coordonnées sont entre parenthèse) :

```
Racine (0)
-- Ele 1 (00)
  -- Ele 11 (000)
  -- Ele 12 (001)
-- Ele 2 (01)
  -- Ele 21 (010)
    -- Ele 211 (0100)
  -- Ele 22 (011)
```

Nous avons développé une fonction pour calculer les coordonnées d'une question :

```
QString MainWindow::calculerCoordonnees(QModelIndex index)
{
    QString coord;
    coord = QString::number(index.row());
    QModelIndex currentIndex = index.parent();
    while(currentIndex != QModelIndex())
    {
        coord = QString::number(currentIndex.row()) + coord;
        currentIndex = currentIndex.parent();
    }
}
```

```

    return coord;
}

```

Cette fonction permet donc, à partir d'un QModelIndex, de calculer ses coordonnées. On peut ensuite, grâce à la méthode "value()" de la QMap, récupérer la valeur (l'objet Question *) associée à la clé (les coordonnées).

Une des difficultés a été de garder la cohérence entre ce qui est affiché dans les treeview et ce qui se trouve en mémoire.

4.4 Visualisation des médias

Les questions, réponses et résultats de notre arborescence peuvent être accompagnés de médias pour aider l'utilisateur à se déterminer. Les 3 principaux formats de médias retenus sont les images, les sons et les vidéos. Nous avons donc développé 3 méthodes pour afficher ces médias.

4.4.1 Visualisation des images

Le média le plus utilisé et donc le plus important sera l'image. Pour intégrer une image dans une application Qt, il nous faut posséder son chemin (le "path"). Ce chemin est stocké dans les objets "Media" (attribut "path"). A partir de ce chemin, on peut créer un objet QImage. Pour vérifier que le chemin est correct et qu'il existe bien un fichier image à cet emplacement, il faut s'assurer que l'objet QImage obtenu n'est pas NULL. Ensuite, on peut éventuellement redimensionner notre image à la bonne taille avant de la convertir en QPixmap. Il suffit désormais d'utiliser cet objet QPixmap dans un QLabel pour voir s'afficher notre image dans l'interface de l'application PC.

```

QImage * myImg = new QImage("images/" + txtCurIdx);

if(myImg->isNull() != true)
{
    QImage myScaledImg = myImg->scaled(QSize(250, 250), Qt::KeepAspectRatio);

    QPixmap * img = new QPixmap();
    img->convertFromImage(myScaledImg, Qt::AutoColor);

    ui->labelImage1->setPixmap(*img);
}

```

[]

Figure 4.4: Code permettant d'afficher une image

4.4.2 Lecteur audio

L'objectif était de pouvoir lire plusieurs types de médias et donc, parmi eux, se trouvaient les fichiers audio. Nous avons donc choisi de créer un lecteur intégré à l'application pour ne pas avoir à dépendre du système d'exploitation sur lequel on l'installera. Le lecteur est très simple. Il est constitué de 6 éléments :

- Un bouton PLAY,
- Un bouton PAUSE,
- Un bouton STOP,

- Une barre de défilement du temps,
- L'affichage numérique du temps actuel du média,
- Une barre de volume.

Après quelques recherches sur les librairies Qt permettant de gérer mes fichiers audio, nous avons choisi d'utiliser Phonon, une librairie qui permettait à la fois de gérer les fichiers audio et vidéo.

Pour programmer notre lecteur audio nous utiliseront différents objets de la librairie. Tout d'abord le "MediaObject" qui permet de recueillir toutes les informations du média entré en paramètre. Il sera donc l'origine de notre lecteur. Nous aurons ensuite les objets "SeekSlider" et "VolumeSlider" permettant de gérer, pour la première, le déroulement du temps grâce à une barre avec un curseur en mouvement et, pour la seconde, gérer le volume grâce à une barre similaire.

Après avoir relié ces trois objets entre eux dans le constructeur de notre classe "mainwindow", il nous suffit de connecter grâce à la méthode "connect()" l'action "clicked" des boutons aux actions "play()", "pause()" et "stop()" qui respectivement mettent le média en lecture, pause ou l'arrêtent en le réinitialisant au début. Il est aussi nécessaire de connecter le signal de changement de média, "currentSourceChanged(Phonon::MediaSource)" où "MediaSource" est le média entré en paramètre dans le "MediaObject", avec le slot "changerSourceVolume()", qui permet de modifier le volume du média en fonction de la barre de volume, ainsi que "changerSourceAvancement()" permettant de modifier la source de la barre d'avancement.

Enfin, il faut utiliser le signal `à tick()` qui permet d'envoyer un signal à un temps régulier pour pouvoir actualiser le temps du média sur la barre ainsi que sur l'affichage LCD qui nous permet de visualiser le temps de la vidéo.

Nous avons laissé le temps d'actualisation à sa valeur par défaut qui est d'une seconde soit 1000 millisecondes car le paramètre d'entrée de cette fonction doit être donné dans cette unité de temps. Grâce à ce signal envoyé toutes les secondes, nous pourrons effectuer le calcul du temps qui s'est écoulé dans le média grâce à une soustraction, effectuée dans la fonction "changerTemps()" entre le temps total, donné par la fonction "totalTime()" du MediaObject, et la durée restante, donnée par la fonction "remainingTime()" de ce même objet.

Il ne nous reste plus qu'à faire un calcul pour passer cette durée en heures, minutes et secondes tout en rentrant ces données dans un objet QTime. On affiche alors le résultat dans le "LCD Number".

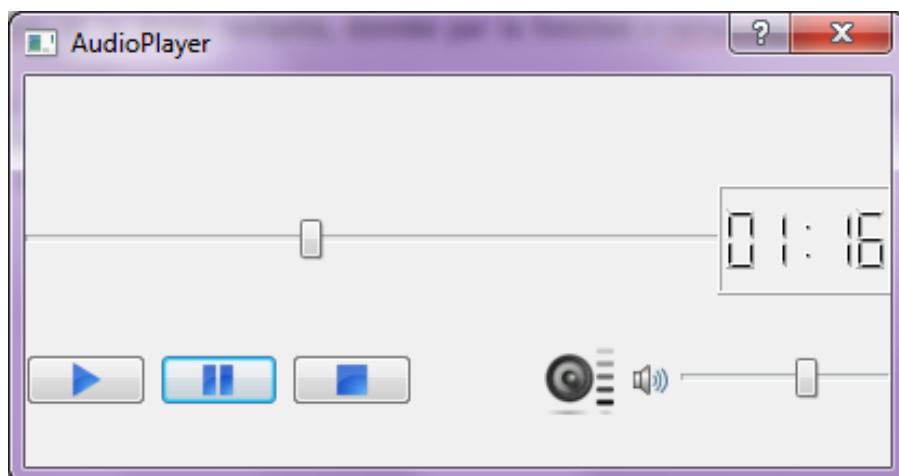


Figure 4.5: Capture d'écran du lecteur audio

4.4.3 Lecteur vidéo

Tout comme le lecteur audio, le lecteur vidéo devait comporter des boutons ainsi que des barres pour pouvoir gérer le média en lecture. Dans le cadre ce lecteur, nous avons simplement ajouté la gestion de la vidéo grâce à un écran pour visualiser les images.

Le lecteur vidéo gère les actions sur les médias de la même façon que le lecteur audio. Cependant, il faut lui ajouter la gestion des images.

Pour cela, nous utilisons un objet "Phonon::VideoWidget". Celui-ci permet de créer une zone de visualisation de la vidéo. Il faudra ensuite relier notre MediaObject à notre VideoWidget ainsi qu'à notre sortie audio (AudioOutput) grâce à la fonction "Phonon::createPath()" prenant en paramètre le MediaObject ainsi que la sortie. Il faudra donc utiliser cette fonction deux fois dans notre cas, une fois pour le son et une fois pour l'image.

Il est ensuite possible de paramétrier son lecteur vidéo avec différents paramètres pour modifier l'aspect de la zone l'affichage comme ajouter un filtre sur l'image. Dans notre cas, nous avons choisi de garder le filtre normal de toute notre fenêtre.

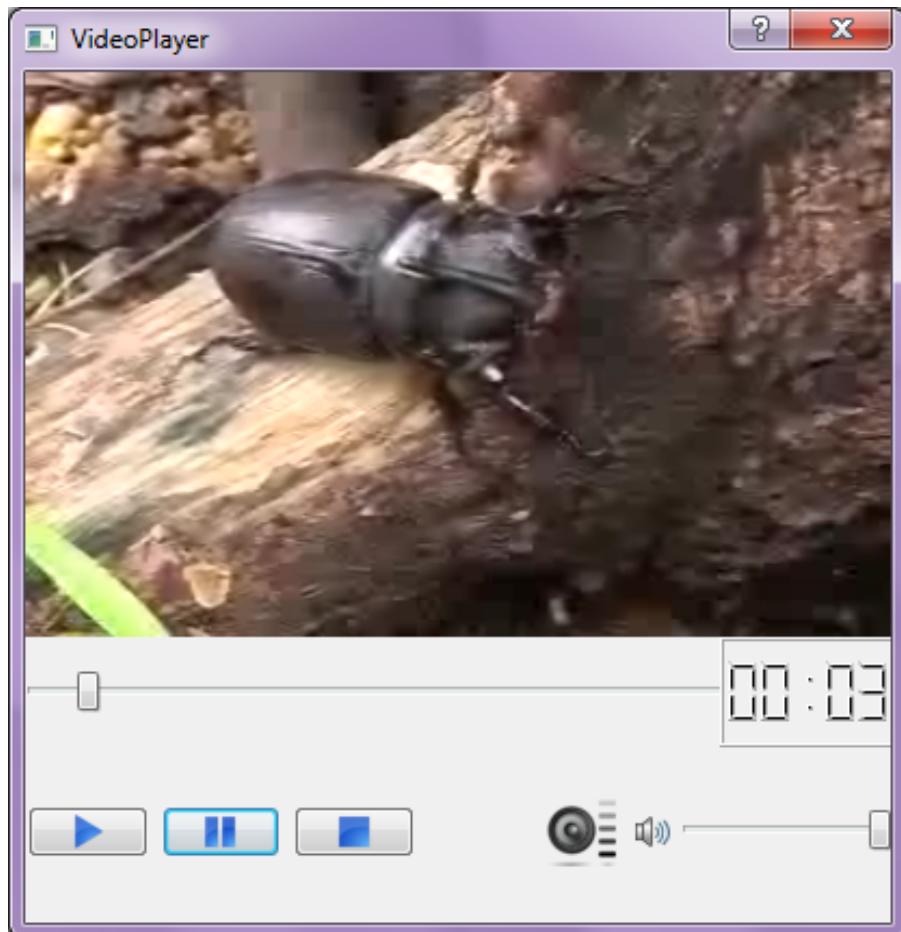


Figure 4.6: Capture d'écran du lecteur vidéo

4.5 Algorithme de coloration des questions

L'objectif de cette fonction était de modifier la petite bulle de couleur permettant de savoir s'il y a un nombre d'informations suffisantes ou non. Notre code couleur fut le suivant :

- Rouge : aucune information,
- Jaune : une information ou média,
- Vert : au moins deux informations ou médias.

L'algorithme de ce programme est directement inséré dans la fonction "peuplerListeQuestionsXML()" qui permet de ressortir toute la liste des questions.

Dès qu'une question est récupérée du fichier XML, nous regardons le nombre de médias et informations insérés dans sa "listeMedia". L'objet "listeMedia" étant une liste chaînée, il nous suffit de récupérer sa taille pour déterminer son nombre d'éléments.

Ensuite nous effectuons des tests suivant le nombre d'éléments détectés puis de modifier la couleur du "QStandardItem" correspondant à la question courante.

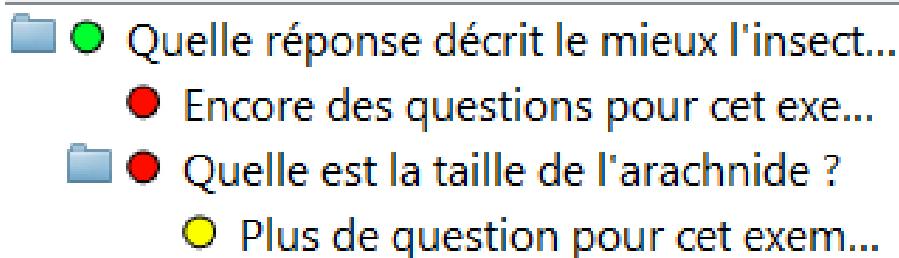


Figure 4.7: Coloration des questions selon la quantité d'informations présentes

Application mobile

La tablette est l'outil utilisé par les biologistes de terrain. Elle doit permettre d'exploiter le fichier XML conçu via le logiciel bureau. Un utilisateur doit pouvoir, via une série de questions, associer l'insecte qu'il observe sous la caméra avec une morpho-espèce de notre base de données. Pour cela il dispose d'aide textuelle, ou visuelle lui permettant de repérer des points clés sur les insectes permettant leurs identifications. L'application tablette est destinée à des personnes non scientifiques. Ceci a été très important pour nous puisqu'il a fallu concevoir une interface graphique adaptée simple, sans terme technique particulier et tout en gardant en tête que la personne devait pouvoir se retrouver facilement sans aucune connaissance pré requises.

Les croquis de base de l'interface d'identification ont constamment évolué au cours du développement de l'application. En effet, nos manipulations fréquentes nous ont permis de nous rendre compte de certaines choses non pratiques notamment en temps de réponse, par exemple l'acquisition de la caméra. Les remarques de l'équipe INNOPHYT ainsi que de M. Venturini, nous ont aussi poussées à constamment modifier notre maquette.

Voici les principales fonctionnalités de l'application tablette :

- Gestion des utilisateurs, des campagnes, des parcelles et des pièges
- Identification de l'insecte via une série de questions pour guider le biologiste
- Identification rapide via l'image de l'insecte
- Gestion des erreurs et des incompréhensions du biologiste
- Création d'un rapport au format CSV exploitable par INNOPHYT

5.1 Fonctionnalités

La partie identification de la tablette possède un certain nombre de fonctionnalités requises que nous allons détailler.

L'utilisateur voit directement dans une zone assez grande le flux vidéo en direct de la caméra arrière de l'appareil. Par une simple pression sur cette zone celui-ci sauvegarde l'image, pour pouvoir la comparer à d'autres photos par la suite. Juste au-dessus de cette zone, ce trouve la question de base de l'arbre XML avec une aide visuelle et textuelle au besoin. Sur la droite de l'application se situe la zone des réponses. Chaque réponse peut contenir une galerie de média, pouvant être scrollé de droite à gauche et comparer à l'image sauvegardée via un long clique sur une des images de la galerie. Si l'espace occupé verticalement par les réponses est trop grand la zone sera rendue scrollable et pourra contenir autant de réponses que l'on souhaite. Le changement de questions et donc l'avancer dans l'arbre d'identification s'effectue en cochant une des réponses. Dans le cas où la question suivante de conviendrait pas à l'utilisateur, un historique permet de revenir en arrière sur ses choix. Lors de la mise en veille ou en arrière-plan de l'application celle-ci est sauvegardée dans son état et sera restaurée avec l'historique dans l'état dans lequel elle était. On peut quitter et reprendre l'identification à son aise. Lorsque nous sommes dans la partie identification

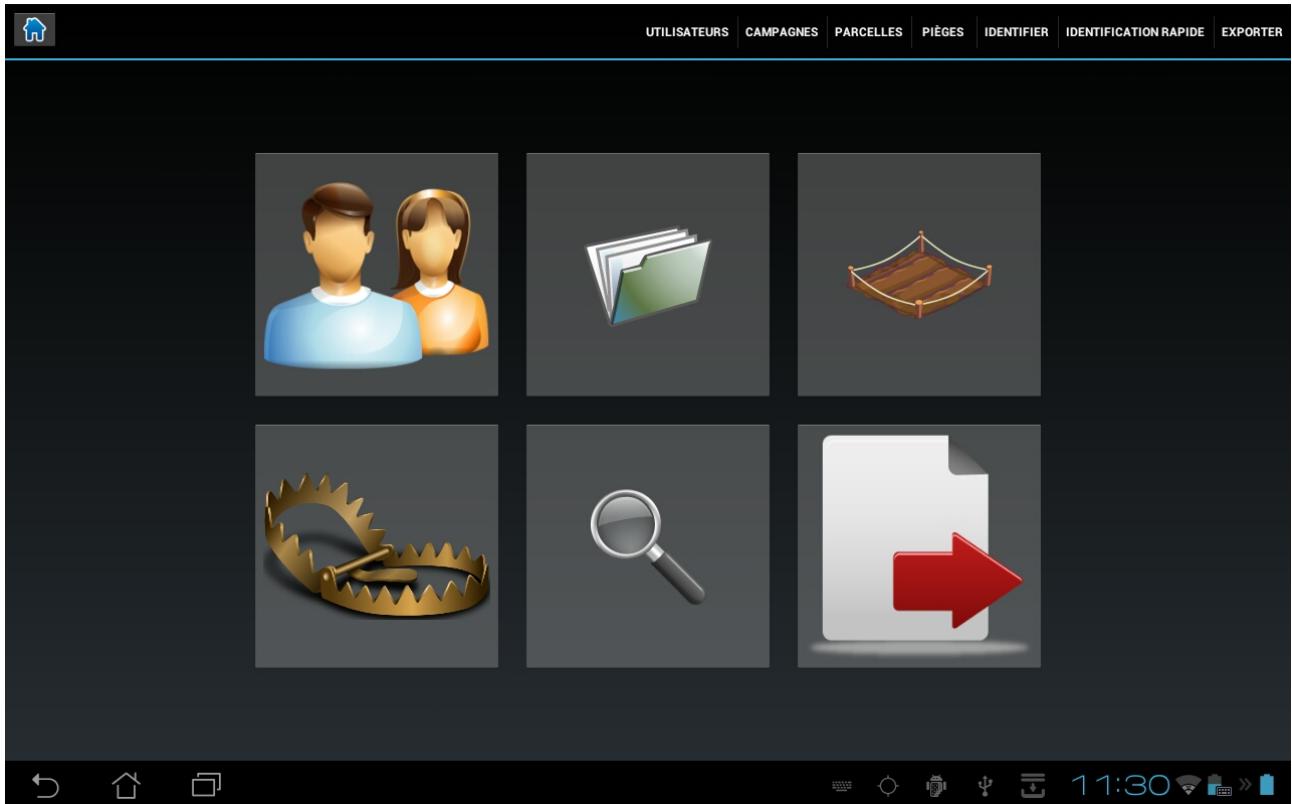


Figure 5.1: Menu d'accueil de l'application mobile

un bouton d'alerte triangulaire orange apparaît dans la barre d'action, permettant à tout moment de signaler que nous sommes bloqués dans le processus d'identification. Cela génère un rapport contenant des informations utiles comme un screen de l'écran utilisateur ou ses commentaires, pour analyse ultérieure.

Ceci est une présentation sommaire des fonctionnalités offertes, dans la prochaine partie nous allons rentrer plus en détail dans la partie technique.

5.2 La zone d'affichage

L'application possède deux zones importantes, l'action barre contenant un menu fixe, et le reste de l'écran en dessous qui change en fonction du besoin que l'on a. Cette dernière zone est gérée via des fragments. Un fragment est un bout d'interface graphique qui a son propre cycle de vie et qui peut être réutilisé et dupliqué en plusieurs objets différents. Ceci est plus complexe à gérer qu'un simple changement d'interface, mais est plus optimisé, stable, propre et très fortement recommandé par la documentation Android pour les applications sérieuses.

Voici quelques conseils et considérations sur l'utilisation des fragments :

1. Lors du `onAttach` conserver dans votre fragment une référence sur l'activity. Cela est essentiel notamment pour retrouver le `FragmentManager` de l'activity dans notre `QuestionFragment`.
2. On ne peut accéder aux éléments définie dans un fichier de layout XML personnalisé que lors du `onActivityCreated` pas avant, difficilement après.
3. La veille via le bouton retour de l'application détruit le fragment contrairement à celle du menu home qui la met en pause et au bouton d'alimentation qui la met en position stop.

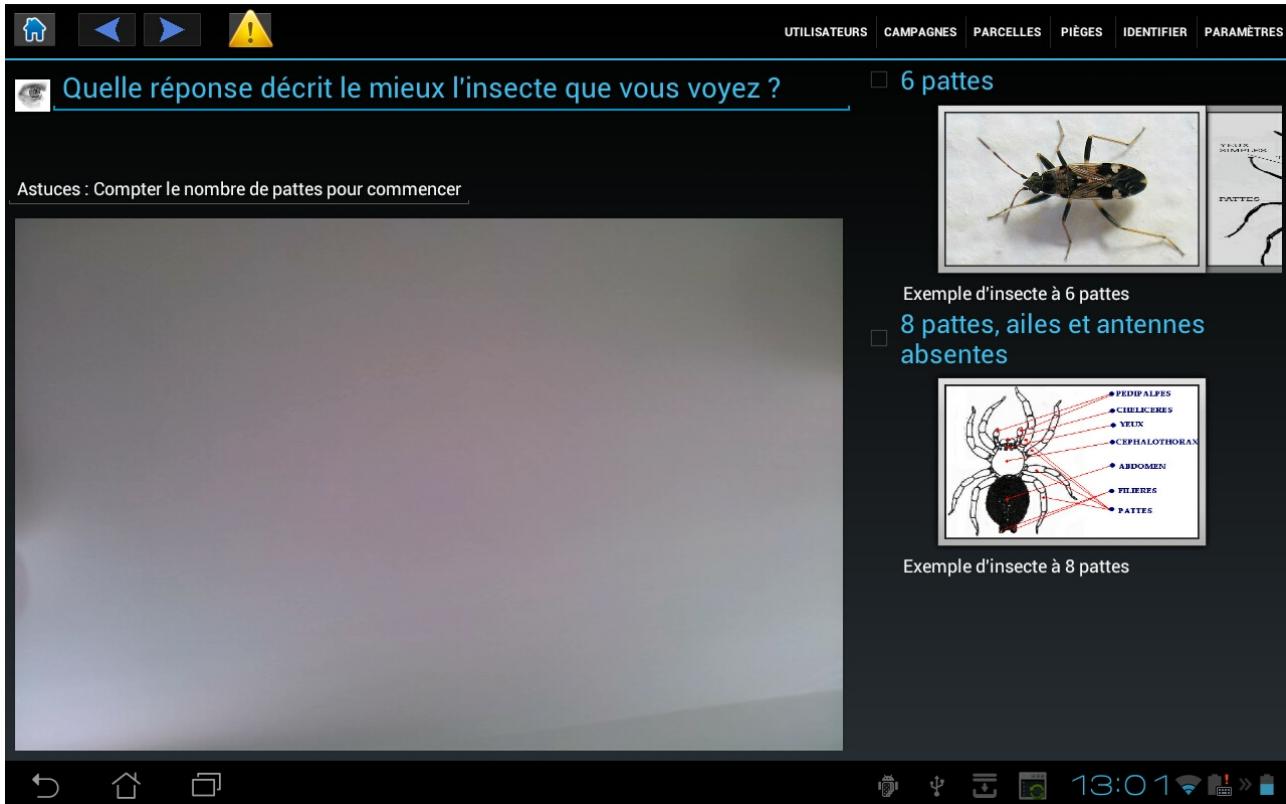


Figure 5.2: Interface d'identification d'un insecte - application mobile

4. Un système de log complet à était mis en place il est très utile pour savoir ce qu'il se passe, penser à l'utiliser et le compléter au besoin.
5. Lors de l'ajout de plusieurs fragments à la suite via le code, il faut bien penser qu'un fragment doit obligatoirement être contenu dans un layout.

Au moment où l'on accède à l'interface d'identification nous chargeons dans l'espace principale le Fragment QuestionFragment. Celui-ci est responsable de beaucoup de chose : - Il récupère l'accès à la camera - Lit la question dont il a besoin dans le XML - Gère l'historique des questions parcourues.

5.3 La caméra

La gestion de la caméra est assez délicate. En effet, si l'on passe outre l'intent de base fournie par Android, il faut tout redévelopper à la main. Ce que nous avons fait, il faut garder à l'esprit que la zone d'affichage quand elle est mise en pause ou détruite doit rendre l'accès à la caméra sinon, elle n'arrivera pas à la récupérer lors de sa sortie du mode pause. Et plus important aucune autre application, dont l'appareil photo de la tablette, ne pourront plus utiliser la caméra. **En cas de plantage de l'application sur l'acquisition de la caméra lors de test, killer l'application puis attendre un petit moment et essayer de lancer l'appareil photo de la tablette si celle-ci y arrive, reprendre les tests, sinon il faut redémarrer la tablette avant de continuer.** L'exemple fourni dans la documentation Android ne nous suffisait pas et faisait fréquemment planter l'application. Pour éviter cela, nous avons ajouté un booléen couplé avec l'acquisition de la caméra pour ne pas pouvoir la rendre ou la capturer deux fois au système, ce qui causait ces crashes. Une fois ce problème d'acquisition résolu nous avons pu nous consacrer à la libération de la caméra. Quel que soit la mise en veille ou action tuant notre fragment ils passeront tous par l'étape en onPause du cycle de vie du fragment, c'est pourquoi la libération de la caméra s'effectue

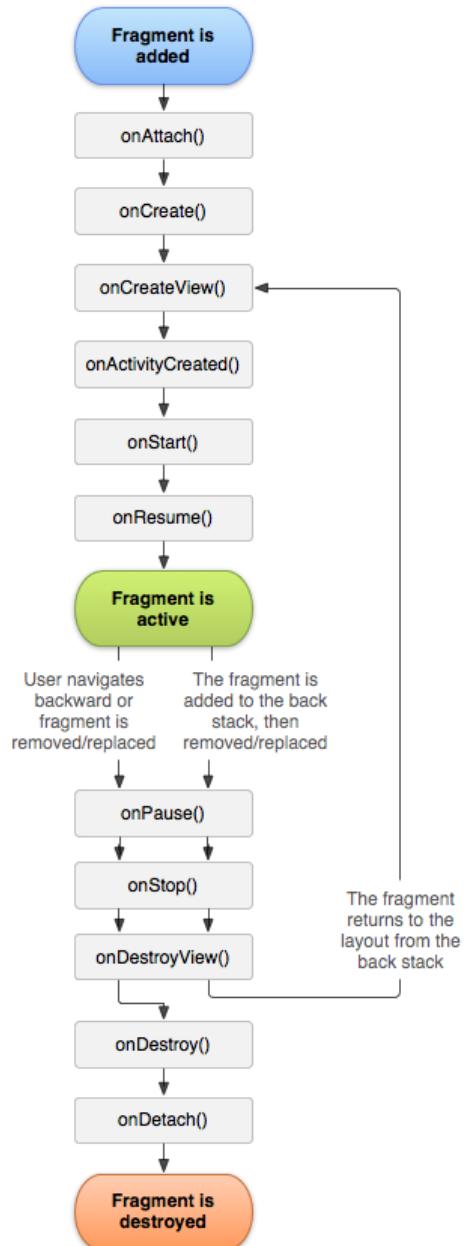


Figure 5.3: Schéma explicatif du fonctionnement des fragments

à cet endroit. Même raisonnement pour le retour de mise en veille et la création du fragment ils passent tous par l'étape onResume c'est donc là que nous faisons notre acquisition de la caméra.

La caméra récupérée est maintenant récupérée et affichée, seulement l'image n'était pas bonne du tout, et cela est normale puisque nous gérons la caméra d'un point de vue bas niveau il faut lui spécifier dans ses paramètres tous les modules que nous voulons activer. Cela s'effectue au niveau de la fonction d'acquisition de la caméra. Nous avons activé trois modules :

1. La balance automatique des blancs
2. La mise au point automatique
3. Et l'antibanding qui essaye d'atténuer l'effet de bande horizontale, comme lorsque qu'on filme un écran de télé avec un caméscope.

```

public Camera getCameraInstance() //Retourne null si la camera n'a pas pu etre acquise
{
    if( !aquis ) //Si la camera n'est pas deja acquise
    {
        Camera c = null;

        try
        {
            c = Camera.open(); //Obtention de la camera de base

            Camera.Parameters params = c.getParameters(); //Activation des paramètres de base de la caméra
            params.setWhiteBalance( Camera.Parameters.WHITE_BALANCE_AUTO );
            params.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
            params.setAntibanding( Camera.Parameters.ANTIBANDING_AUTO );
            c.setParameters(params);

            aquis = true;
            mCamera.startPreview();
        }
        catch (Exception e)
        {
            Log.d("Camera", "la camera n'est pas accessible ou n'existe pas");
        }
        return c;
    }
    else
    {
        Log.d("Camera", "getCameraInstance camera non relaché" );
        return mCamera;
    }
}

```

Figure 5.4: Code permettant d'éviter les problèmes liés à la caméra

Maintenant reste à expliquer un point qui fût particulièrement problématique. Lors de la mise en veille, via la touche power, de la tablette et de sa remise en route nous retrouvions l'application dans l'exact même état qu'avant à l'exception de la zone vidéo qui était figée sur la dernière image, vue avant la mise en veille. Nous avons cherchons longtemps, en partant notamment sur des fausses pistes comme une mauvaise acquisition ou libération de la caméra, finalement cela venait du bout de code donner par Android pour commencer. En effet, nous devons définir un nouveau type de zone pour pouvoir afficher le flux de la caméra sur cette zone. Ce bout de code nous est fourni par la documentation Android et nous l'avons considéré trop longtemps comme acquis sans y regarder de plus près. Et lorsque nous l'avons fait, nous nous sommes rendu compte qu'il manquait tout simplement une fonction de rafraîchissement de cette zone. Pourquoi il y a besoin de rafraîchir la zone uniquement lors de la mise en veille de la tablette et non lors de la mise en veille de l'application cela reste pour le moment un mystère.

5.4 Les bitmaps

Toutes les images affichées dynamiquement dans l'interface graphique sont des bitmaps. Cela soulève plusieurs problèmes. La taille des images chargées provoque quasi instantanément un dépassement mémoire, provoquant un segfault lors de l'affichage de l'image suivante. Ce problème nous a demandé beaucoup de travail, car il est plus ou moins rapide à survenir. Au départ le chargement d'une seule image provoquait l'erreur, puis au bout de cinq ou plus. Nous avons mis en place deux procédures pour éviter ce problème. Nous libérons dès que possible les objets de type bitmap pour éviter de surcharger le système. Cela se fait via l'appel à la fonction recycle des objets de type bitmap. Conséquence directe de cette procédure nous avons dû rajouter dans la classe ImageAdapter un déstructeur `~`, ce qui normalement ne se fait pas en java.

```

public void update( Camera cam )
{
    mCamera = cam;

    try
    {
        mCamera.setPreviewDisplay(mHolder);
        mCamera.startPreview();
    }
    catch (Exception e)
    {
        Log.d("tag", "Error starting camera preview: " + e.getMessage());
    }
}

```

Figure 5.5: Mise à jour de l'objet caméra et rafraîchissement de la zone de visualisation de la classe CameraPreview

```

public void finalize()
{
    for( int i = 0; i < bitmaps.length; i++ )
    {
        bitmaps[i].recycle();
    }
}

```

Figure 5.6: Destructeur de la classe ImageAdapter

Nous avons aussi dû demander au système de réduire la taille des images enregistrées en mémoire d'un facteur de 1/8 (valeur recommandée par plusieurs utilisateurs ayant rencontrés le même problème et confirmé par nos soins) diminuant ainsi grandement les chances de dépassement mémoire.

5.5 L'historique

La gestion de l'historique des questions n'a pas posé de problème mais nécessite d'être expliquée. Au démarrage du fragment, le parsing du fichier XML est lancé, il nous renvoie la première question que nous stockons dans la classe sous le nom de currentQuestion. Une fois cette question récupérée nous appelons la fonction changeUI qui adapte l'interface graphique du fragment à la currentQuestion. Le but du jeu est donc de changer en fonction de la position dans l'historique la currentQuestion et de rappeler changeUI. Pour cela nous avons trois éléments :

1. Un vecteur contenant toutes les questions vues par l'utilisateur listQuestion
2. Un vecteur contenant toutes les réponses choisies par l'utilisateur listReponseChoisi
3. Un int navigation qui indique la position dans le vecteur listQuestion de l'historique

Nous pouvons en déduire que lorsque navigation = listQuestion.size()-1 nous sommes à la fin du vecteur listQuestion et donc nous sommes sur la dernière question vue par l'utilisateur, une question à laquelle il n'a pas encore répondu. Nous devons donc activer la flèche de retour en arrière tant que navigation != listQuestion.size()-1 et navigation >= 0. Si navigation atteint 0 c'est que nous sommes au début de l'historique il faut désactiver la flèche de retour dans l'historique. Même principe pour la flèche suivant de l'action barre, si navigation >= 0 et navigation != listQuestion.size()-1 alors il faut activer la flèche sinon

la désactiver.

Pour obtenir la question suivante de la question actuelle il suffit d'utiliser la classe ReacherQR responsable du parsing du fichier XML. Dans le cas où la question n'est pas une question finale, et mène donc à une autre question cela se résume à :

```
currentQuestion = rechercher.findQuestionById( reponse.getIdQuestionSuivante() );
listQuestion.add( currentQuestion );
```

Figure 5.7: Code pour passer à la question suivante

Il faut bien penser à gérer le cas où l'utilisateur revient en arrière dans l'historique et fait un choix différent. Dans ce cas il faut supprimer toute la suite de l'historique à partir du point changé.

Maintenant il faut pouvoir retrouver et cocher la checkbox de l'ancienne réponse que l'utilisateur a choisie, parmi toutes les réponses que possède la question. Pour cela nous conservons toutes les réponses à la currentQuestion dans un vecteur listReponseFragment. Cela nous permet de le parcourir et par comparaison de retrouver et de cocher la checkbox de l'ancienne réponse cochée.

5.6 La gestion des projets

5.6.1 D'un point de vue utilisateur

La tablette peut être utilisée par plusieurs biologistes, dans le logiciel, l'utilisateur est identifié par son nom. L'application ne supporte qu'un utilisateur en même temps. Les biologistes travaillent sur différentes campagnes : une campagne est un projet défini dans le temps et dans l'espace. Cette localisation se fait par l'intermédiaire des parcelles où les pièges sont posés. Pour chaque piège, plusieurs insectes sont trouvables et comptables.

Pour pouvoir utiliser l'application, l'utilisateur doit donc s'identifier en indiquant son nom. Après il peut créer, sélectionner, éditer ou supprimer ses campagnes. Il possède les mêmes fonctionnalités pour les parcelles et les pièges.

On observe une structure arborescente où l'utilisateur est la racine. Au deuxième étage se trouve les campagnes, au troisième les parcelles, au quatrième les pièges et pour finir les insectes trouvés.

Pour se faire, le biologiste est amené à se déplacer dans différentes vues pour sélectionner le piège sur lequel il travaille. Chaque vue (campagne, parcelle, piège) demande le même type d'informations :

- Un nom
- Un descriptif
- Une date de début et de fin
- Une adresse
- Une position GPS

Les pièges exceptés, seul le nom est obligatoire, les autres informations sont disponibles pour permettre au biologiste d'être le plus précis sur son travail. La particularité des pièges vient du fait qui doivent être limité dans le temps afin de pouvoir suivre l'évolution de la quantité d'insecte qu'ils capturent. C'est pour quoi, il faut obligatoirement que la date de fin soit indiquée.

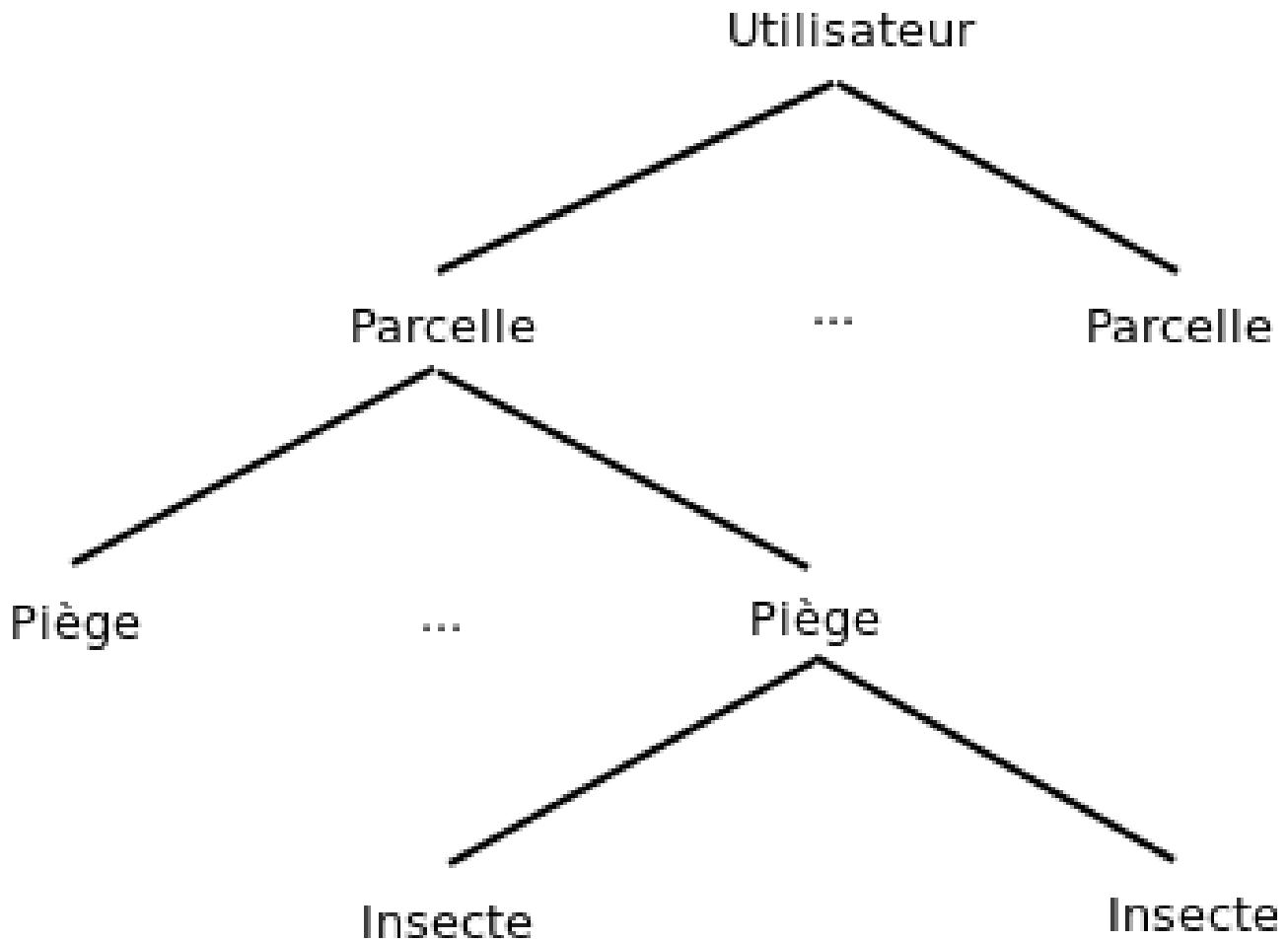


Figure 5.8: Structure arborescente de la gestion des utilisateurs

Dans ces différentes vues, les champs dates peuvent être renseignées avec le clavier ou avec un calendrier qui apparaît lors d'un double clic sur la case date.

Le bouton GPS permet de récupérer la latitude et longitude actuelles, ces informations ne sont disponibles que s'il y a suffisamment de satellites connus par l'appareil ; cela marche très bien en extérieur. En cas d'échec un message apparaît.

Les images ci-dessous vous montrent les vues pour la gestion des campagnes, des parcelles et des pièges. On peut constater leur grande ressemblance. Cela permet à l'utilisateur de n'avoir qu'un seul type d'interface à prendre en compte, cela vise à le rassurer.

5.6.2 Structure du logiciel

Les informations sur les utilisateurs et les projets sont enregistrées dans une base de données SQLite. Plusieurs tables ont été créées et les clefs étrangères font la liaison entre elles. Le diagramme de la figure 5.13 montre leur interaction.

On retrouve une table utilisateur, définie par un identifiant et un nom. Le nom doit être unique pour éviter les doublons et les problèmes de connexions que cela poserait.

La table campagne est définie par un identifiant, un nom, des dates de début et de fin, une description, une adresse et une position GPS définie par une latitude et une longitude. La relation avec la table utilisateur est faite avec une clef étrangère qui est l'identifiant de l'utilisateur possédant la campagne. Cette table

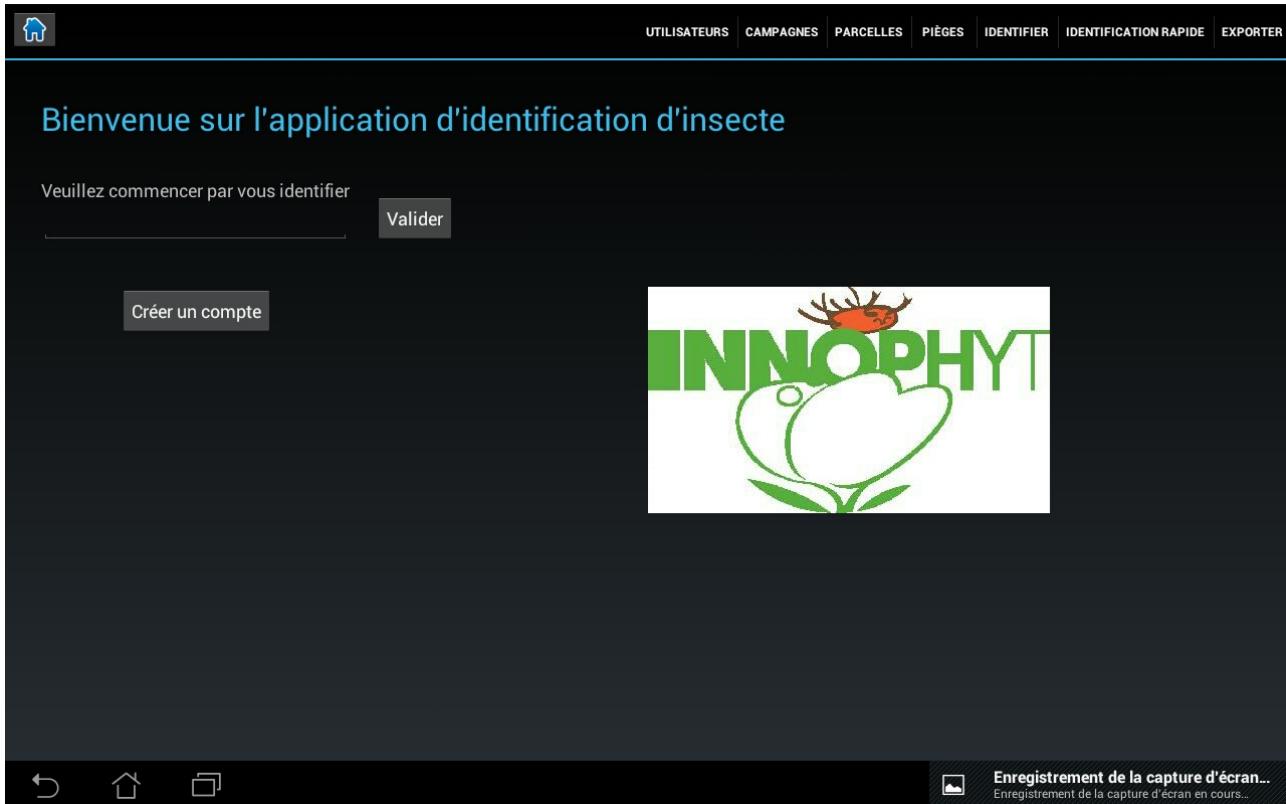


Figure 5.9: Interface d'identification d'un utilisateur

possède une contrainte d'unicité sur son nom et l'identifiant de l'utilisateur, de manière à s'assurer qu'un biologiste ne puisse pas faire de doublon et que son collègue puisse tout de même créer une campagne de même nom.

Les tables parcelle et piège sont construites en suivant le même modèle. Mis à part le principe d'unicité de la table piège qui est étendu à la date de fin pour que les relevés puissent être suivis dans le temps. Pour la dernière table nommée Récolte, le principe de clef étrangère reste le même. Le terme récolte désigne tous les insectes comptés et enregistrés.

Le diagramme de classes se trouvant en figure 5.14 montre les interactions entre la base de données et l'application.

La classe GestionBDD permet la création et la réinitialisation des tables de la base de données. Elle est utilisée par les autres classes de gestion pour se connecter à celle-ci.

Les classes UtilisateurBDD, CampagneBDD, ParcalleBDD, PiegeBDD, et RecolteBDD permettent l'ouverture et la fermeture de la base de données, l'insertion, la suppression et la modification ainsi que la lecture d'éléments de la table. Respectivement, elles gèrent des utilisateurs, des campagnes, des parcelles, des pièges et des récoltes. Les classes Utilisateur, Campagne, Parcalle, Piège et Récolte sont présentes pour instancier ces informations. Ce sont ces objets qui sont utilisés pour communiquer avec le gestionnaire de base de données.

Les vues présentées précédemment demandent, modifient, et renvoient ces objets au gestionnaire pour lire, écrire modifier et enregistrer des informations. Les vues GestionCampagne, GestionParcalle et GestionPiege gèrent respectivement des liste de Campagne de Parcalle et de Piege ce qui explique les attributs "un à plusieurs" sur les compositions.

Les paramètres de l'application comme les identifiants de l'utilisateur, de la campagne, de la parcelle

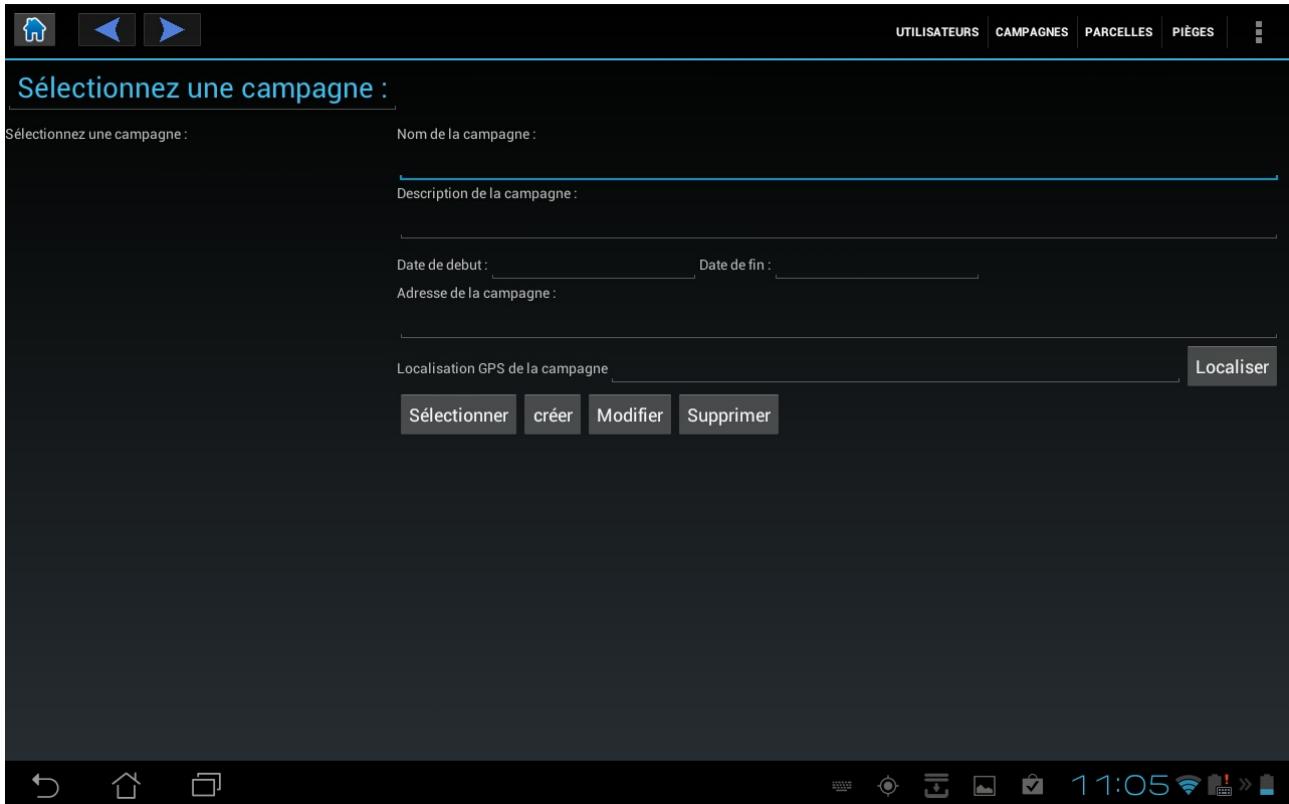


Figure 5.10: Interface de gestion des campagnes

et du piège sélectionné sont enregistrés dans les settings du logiciel. La méthode statique nommée "sécurité" de la classe Securite s'assure que tous les paramètres nécessaires à la vue demandée sont bien présents dans les settings. Si ce n'est pas le cas; l'utilisateur est redirigé vers la vue nécessaire pour combler le manque.

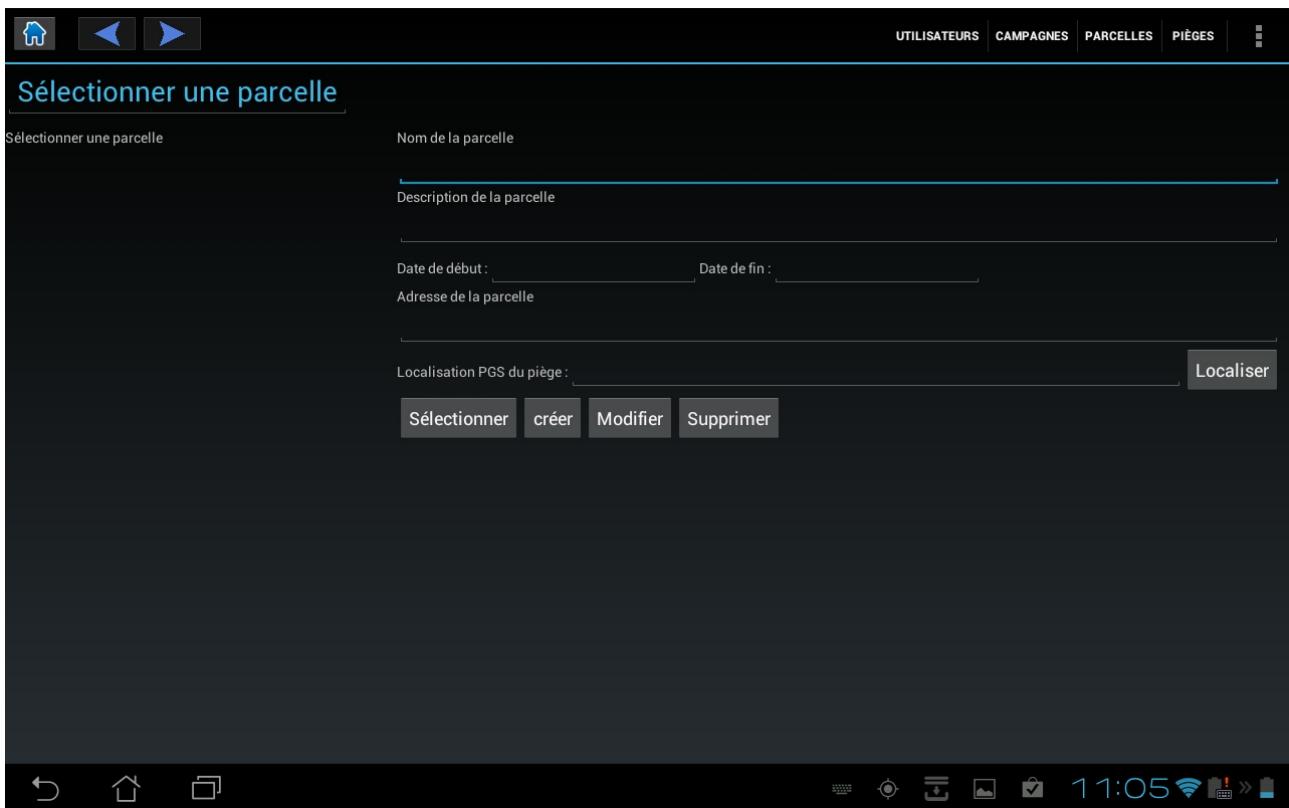


Figure 5.11: Interface de gestion des parcelles

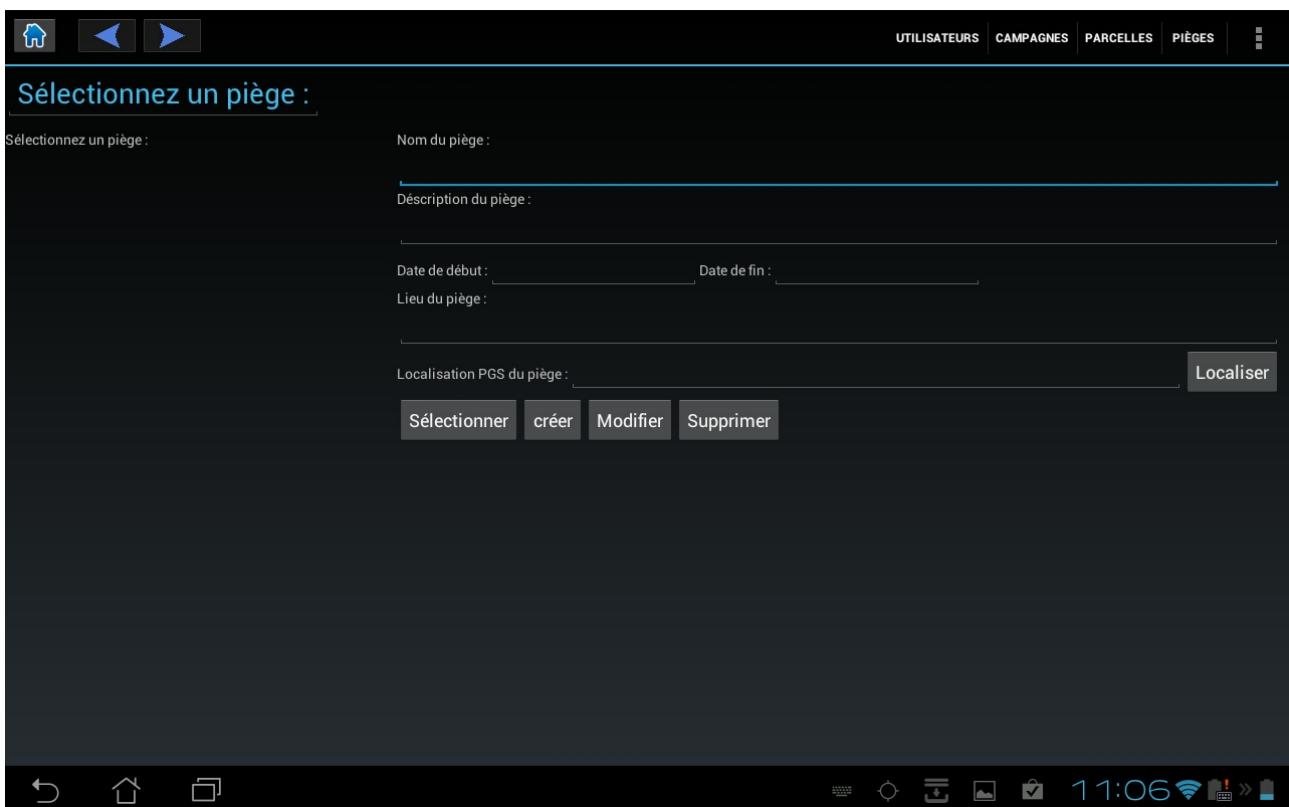


Figure 5.12: Interface de gestion des pièges

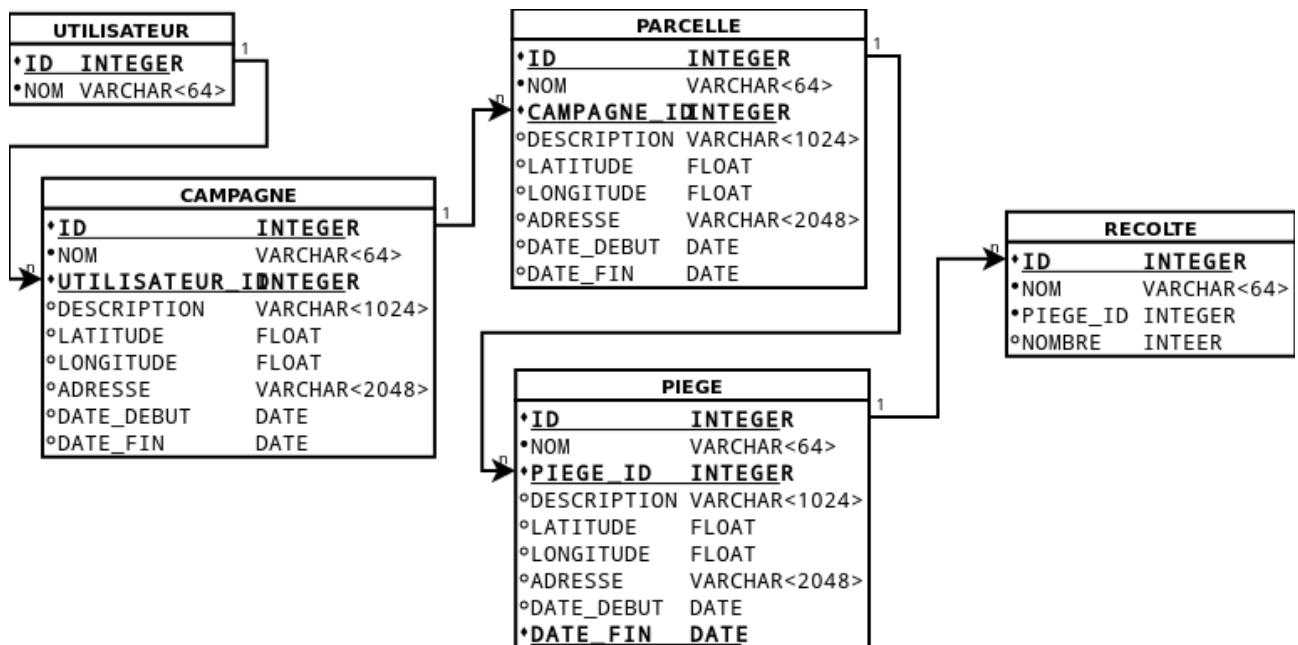


Figure 5.13: Modélisation de la base de données

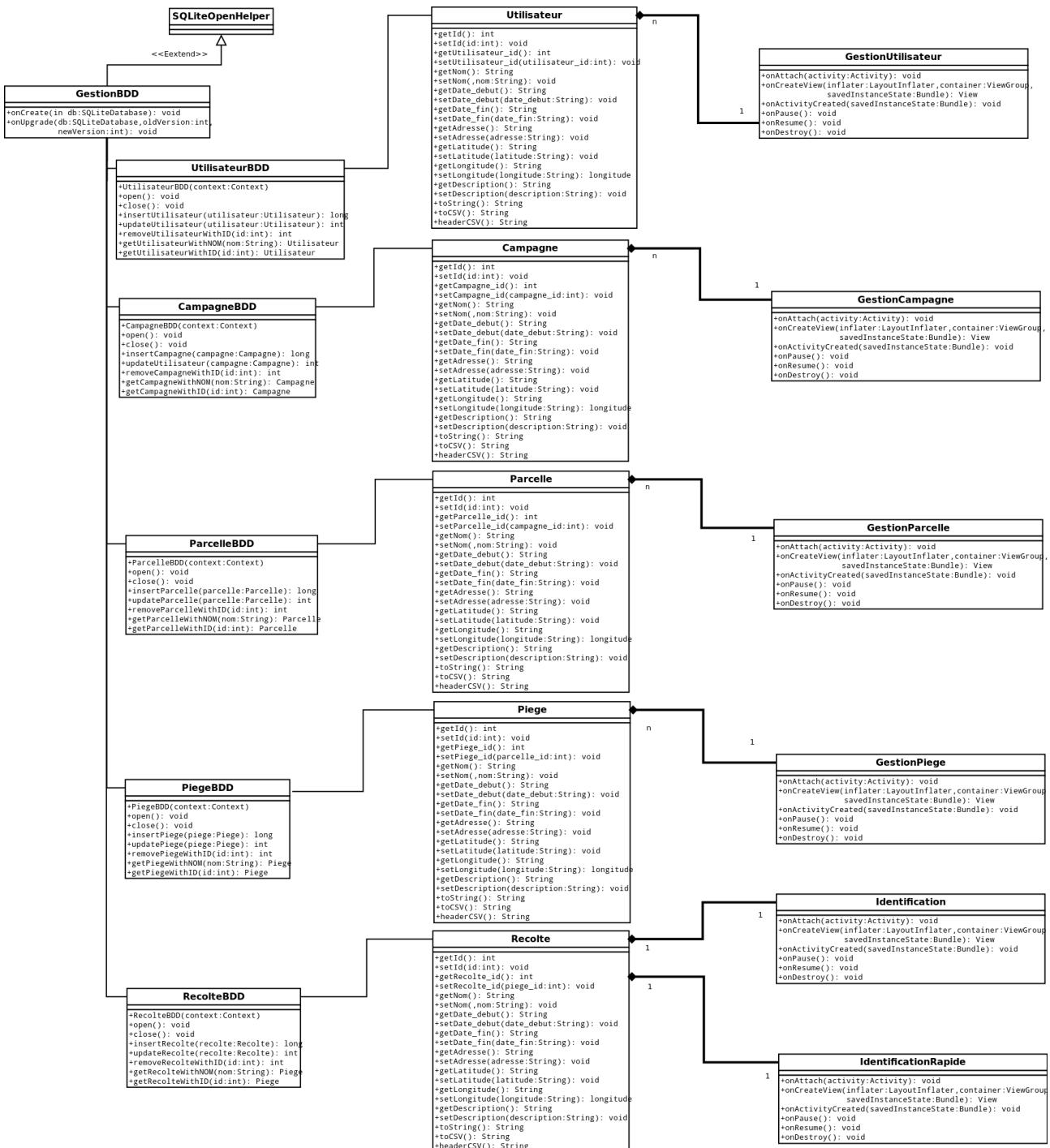


Figure 5.14: Modélisation des interactions entre la BDD et l'application

Aspect gestion de projet

Pour coller au mieux aux attentes du client, nous avons essayé d'organiser le plus souvent possible des réunions de travail avec le client. Ces réunions avaient pour but de discuter des choix d'ergonomie des applications et de présenter notre travail. Cela a permis à l'équipe INNOPHYT de suivre notre travail au plus près, et de relever les erreurs ou les incohérences au plus tôt, afin que nous puissions les corriger au plus vite.

Pour les aspects plus techniques, nous organisions des bilans réguliers avec notre encadrant de projet, Gilles Venturini. Notamment, pour valider les étapes essentielles du projet : le format des données, le contenu du fichier XML, certains points de l'interface, certaines fonctionnalités critiques...

Enfin, au sein même de l'équipe, nous avons essayé d'organiser des points le plus régulièrement possible. Ces points permettaient de partager les difficultés rencontrées, de connaître l'avancement des membres, de savoir ce qu'il restait à faire... Nous faisions un point par semaine, de préférence en début, qui durait en moyenne 10 à 15 minutes.

6.1 Témoignage de l'équipe

Ayant voulu exprimer au mieux l'ambiance de l'équipe ainsi que le ressenti de chacun à propos du projet, nous avons décidé de publier un florilège des témoignages de chaque membre de l'équipe. Ceci est plus complet qu'un bilan synthétique écrit par une seule personne tout en restant assez informel et quelque peu personnel. En complément de ces 8 témoignages, vous trouverez aussi les fiches d'auto-évaluation qui ont été remplis personnellement par chacun (fiche manuscrite qui seront remises lors de la soutenance).

Dans un premier temps, je trouve que ce projet a été correctement mené. Pour notre premier projet avec une équipe de huit personnes, c'est vraiment correct et digne d'un projet de futur ingénieur. On se rend compte des problèmes qui nous arriveront dans notre future vie professionnelle. On remarque que les personnes peuvent s'investir plus ou moins, que les retards sont fréquents... C'est donc une très bonne expérience.

Cependant, je suis un peu déçu du travail que j'ai pu fournir et de ce que j'ai pu apprendre. En effet, la majorité du projet, j'ai dû travailler avec Jérôme. C'est quelqu'un d'intelligent et de bon. Malgré cela, je pensais pouvoir en apprendre de lui mais malheureusement cela ne s'est pas passé ainsi. De ce fait, j'ai du me débrouiller avec ce qu'il faisait quand il le voulait, et je ne pouvais pas vraiment avancer. Il aurait fallu tout reprendre mais c'était trop tard pour tout recommencer. Avec le recul, il aurait fallu que je sois seul sur une partie, et avancer par moi-même.

Je retiens ainsi de cette petite histoire pas mal de choses. En effet, cela nous permet de voir qu'il est souvent difficile de s'adapter à la personne avec qui l'on travaille. Il faut prendre des décisions, rapidement, et elles doivent être bonnes.

Pour conclure, je retiendrais une bonne expérience tout de même étant donné que ce projet m'aura permis de voir comment se passe un projet avec un nombre de personnes assez élevé. Je tiens à féliciter Matthieu qui je trouve a bien géré la situation, ce qui n'est pas évident. Nous avons au final une application fonctionnelle et vraiment agréable.

Martin DEMEULEMEESTER — *Développeur application PC.*

Je ne veux plus toucher de tablette après toutes les difficultés que nous avons rencontrées n Ce fût un vrai challenge et je suis content du résultat que nous avons obtenu. La montée en compétence fût plus longue que je l'espérais, mais reste un bon souvenir maintenant que j'ai une certaine maîtrise.

La constante attention, patience et compréhension de notre chef de projet a été un atout majeur pour le déroulement de notre travail, la bonne ambiance était au rendez-vous.

Julien TERUEL — *Développeur application mobile.*

Ce projet est celui de nouvelles expériences. Il m'a permis de me mettre au Java, plus précisément au développement sous Android, il m'a également permis de voir ce que donne un projet à huit étudiants sur une période discontinue.

C'est aussi un projet que je trouvais ambitieux et concret. Avec du recul, je me rends compte que le temps perdu dans certaines parties du développement provient d'un manque de connaissance initiale sur cette technologie. Mais une fois cette difficulté surmontée et une bonne nuit blanche pour rattraper le retard je trouve le résultat très prometteur. Non seulement il répond aux demandes des clients, mais il peut être une base pour de prochaines évolutions.

Au niveau du projet lui même, je trouve gratifiant de voir le chemin parcouru. De regarder la transition entre les croquis de départ, qui étaient d'avantage basés sur l'imagination que des connaissances fiables, et le résultat qui est relativement proche de l'idée initiale mais surtout, qui cadre avec la volonté d'INNOPHYT. C'est-à-dire d'avoir une application fiable et intuitive. Au niveau des membres du groupe, j'ai plutôt des remerciements à donner que des reproches à formuler. Pour commencer, je tiens à remercier Matthieu, car il a su gérer le projet sans être derrière tout le monde, sans critiquer alors que l'application ne marchait pas. En somme pour la confiance qu'il a su accorder. Il est vrai que certains rappels à l'ordre auraient pu se faire plutôt, mais pour sa première gestion de projet, il s'en est bien sorti et le résultat le prouve. Je tiens également à remercier Julien. Il a travaillé avec moi sur l'application mobile et son implication dans le projet, son aide sur deux parties délicates montrent que l'on peut compter sur lui et qu'il possède l'esprit d'équipe.

Mickael PURET — *Développeur application mobile.*

Ce projet a été très constructif pour moi. Il m'a permis de découvrir de nouvelles technologies comme la gestion de différents types de médias sous Qt grâce à la librairie Phonon. Il m'a aussi fait de réutiliser des connaissances déjà acquises auparavant comme la création de fichiers XML et d'une DTD.

Ce projet nous a mis dans des conditions réelles de projet, avec un client, des objectifs précis, un cahier des charges ainsi qu'une équipe de développement contenant un chef de projet. Celui-ci s'est très bien débrouillé en créant différentes tâches en fonction du cahier des charges et en les répartissant en équipes de deux personnes. Grâce à ces conditions, nous avons aussi pu réaliser les diverses difficultés d'un projet d'équipe comme les délais à respecter ainsi que d'apprendre de nouvelles technologies indispensables au projet.

Simon FAUSSIER — *Développeur application PC.*

Je suis content d'avoir pu travailler dans cette équipe, aucun conflit n'a été relevé, toutes les personnes sont arrivées au bout de leur tâches dans les délais attendu. Je n'ai pas pu m'investir à 100% dans ce projet pour cause personnelle, mais il a été très intéressant et m'a permis, dans ma partie, d'asseoir mes compétences en C++.

Jérôme HEISSLER — *Développeur application PC.*

Durant ce projet d'ingénierie du logiciel, nous avons travaillé dans une équipe de 8 personnes. J'ai bien approfondi mes connaissances par rapport, non seulement au cours de Java, mais aussi au cours de gestion du projet. De plus, cela m'a permis de bien connaître l'importance de l'esprit d'équipe. On a travaillé indépendamment pour une partie, mais on s'est aussi réuni pour trouver des solutions lorsque l'on rencontrait des problèmes, ou pour fusionner les travaux déjà réalisés. On a les membres qui étaient tous responsables. On a aussi le chef du projet qui a bien organisé la progression. Donc, je suis très satisfait d'avoir été dans cette équipe. Par contre, je suis désolé d'avoir causé quelques petits retards durant ce projet à cause de mon inattention, car cela impacte aussi les autres membres. Je vais faire très attention à l'avenir.

Zhengyi LIU — *Développeur application mobile.*

Je pense que notre groupe est très bon. Tous les membres ont leurs caractéristiques et peuvent faire les tâches qui correspondent à leurs compétences. Sur les séances de projet, le taux de présence est proche de 99%. C'est très bon d'avoir fait le projet ensemble.

Pour le chef de projet, il a proposé un bon planning pour notre groupe. Il a bien organisé l'avancement du projet et il a bien communiqué avec l'encadrant, le client et les membres de groupe. Je pense qu'il est un bon chef.

Pour ma part, ce projet m'a permis d'apprendre beaucoup de choses : structure d'un fichier XML, gestion du XML en Java (lecture/écriture) et développement sous Android. Je suis très content d'avoir fait un projet dans ce groupe.

Je regrette de ne pas avoir communiqué suffisamment avec mes camarades français, à cause de mon français et de mon caractère timide. C'est un défaut que je dois modifier pour la suite des mes études en France.

Zheng ZHANG — *Développeur application mobile.*

Ce projet m'a permis d'acquérir une première expérience de "chef de projet". J'ai été confronté aux difficultés de gérer une équipe sur une équipe assez longue (environ 3 mois) et de gérer les demandes parfois fluctuantes du client.

C'était un projet très intéressant. Il m'a permis d'allier du développement d'application "desktop" en C++ et du développement d'application mobile en Java sous Android, domaine et langage qui me passionne. Je suis juste un peu déçu ne pas avoir pu consacrer plus de temps sur la partie mobile, en cause, les retards et les soucis rencontrés sur la partie PC qui m'ont contraint à travailler en particulier sur cette partie-là. Je m'excuse par avance auprès de notre encadrant, et surtout auprès de l'équipe INNOPHYT, de ne pas avoir pu terminer totalement

le projet. Il est fort dommage que l'application PC ne soit pas entièrement fonctionnelle... Au niveau du groupe, nous avions une équipe très intéressante, composée de personnes toutes douées dans un domaine différent. Je suis très content du fort taux de présence lors des séances de projet. Il témoigne de l'intérêt de l'équipe à s'investir dans le projet.

Matthieu ANCERET — *chef de projet*.

6.2 Calendrier prévisionnel et calendrier réel

Les retards ont principalement eu lieu au niveau de la phase de développement (tâche 4 pour le calendrier prévisionnel et tâche 5 pour le calendrier réel). Nous avons commencé cette phase avec une semaine de retard sur le planning à cause de la tâche "Correction du fichier XML et de sa DTD" qui n'était pas prévue à l'origine.

On remarque ensuite que les durées des tâches "Application PC" et "Application mobile" ne correspondent pas entre calendrier prévisionnel et réel.

L'application PC a eu 14 jours de retard. Ce retard s'explique principalement par la durée excessive prise par la tâche "Lecture du fichier XML". Suite à plusieurs difficultés, nous avons perdu beaucoup de temps sur cette tâche. Étant une tâche critique du projet, un retard sur celle-ci a engendré des retards sur les autres tâches qui avaient besoin qu'elle soit terminée (écriture XML par exemple). Le module de visualisation des médias a lui aussi pris beaucoup plus de temps que prévu. Celui-ci étant moins important, son impact sur la faisabilité du projet est moindre (il compte malgré tout dans les 53 jours de développement). Ce retard nous a empêchés de terminer complètement l'application et plusieurs fonctionnalités sont malheureusement manquantes.

L'application mobile a eu 27 jours de retard. Ce retard important est dû au modèle de recherche rapide (identification rapide) qui n'était pas une tâche critique. Ce retard important par rapport au planning prévisionnel est donc à relativiser, et cela n'a absolument pas empêché l'application mobile d'être globalement terminée et fonctionnelle dans les temps.

Futur du projet

Globalement, le projet est utilisable en l'état actuel. Il reste malgré tout plusieurs éléments à terminer ou à compléter, que ce soit par manque de temps ou parce que ces fonctionnalités ne faisaient pas partie du périmètre du projet. En voici une liste détaillée :

- Avec le recul, nous avons remarqué quelques points perfectibles dans l'interface de l'application PC. Ces points nuisent à l'ergonomie de l'application. Notamment, comment symboliser le lien entre une réponse et la question suivante. Nous avons choisi de séparer l'affichage des questions de celui des réponses. Nous avons donc symbolisé la question suivante en affichant son nom lors du clic sur une réponse. Ce nom permettant de se repérer dans l'arborescence des questions.
Représenter clairement et lisiblement une arborescence XML complexe dans une interface est assez compliqué. Nous avons choisi une méthode, peut-être pas la meilleure, car il en existe d'autre.
- Actuellement, nous stockons tous les fichiers des médias dans un seul et unique dossier. Il pourrait être intéressant de trier ces médias dans une arborescence de dossiers (par exemple, reprenant l'arborescence de l'arbre XML ; ou encore par type de médias : un dossier images, un dossier son, un dossier vidéo...). Ceci pourrait apporter plus de souplesse dans le traitement et l'analyse de ces médias.
- Au niveau de l'application mobile, il serait intéressant de pouvoir visualiser les résultats et les statistiques directement sur la tablette, via une interface pratique et adaptée. Actuellement, la seule façon de consulter les résultats est de produire un fichier tableur à partir de tablette, de récupérer ce fichier et de l'ouvrir sur un PC avec un logiciel adapté (Excel, OpenOffice...). Pouvoir les consulter directement sur la tablette permettrait d'éviter cette étape d'exportation et de pouvoir gagner du temps dans le cas où l'utilisateur veut simplement vérifier ou consulter certains résultats intermédiaires en cours d'une campagne d'acquisition.
- Toujours au niveau de l'application mobile, il n'est actuellement pas possible de changer le fichier de données lorsque l'application fonctionne. Il faut arrêter l'application, modifier le fichier XML et son dossier de médias associé et redémarrer l'application. Il serait intéressant de pouvoir changer la base de données (fichier XML + médias) "à la volée", à partir d'un menu de paramètre par exemple. Cela permettrait d'avoir, directement embarquée sur la tablette, plusieurs bases de données (insectes, oiseaux, plantes...) pour étendre les possibilités de l'application.
- Pour rendre l'application tablette presque entièrement indépendante, il faudrait mettre en place un système de mise à jour automatique des bases de données embarquées. Actuellement, une fois la base modifiée via l'application PC, il faut copier manuellement les fichiers de cette nouvelle base sur la tablette. Une idée pourrait être d'installer un serveur sur lequel seraient poussées les mises à jour réalisées par l'application PC et où l'application mobile irait régulièrement chercher les nouvelles informations. Ce mode de fonctionnement permettrait aussi à plusieurs tablettes de récupérer rapidement la dernière version de la base. La seule contrainte étant d'avoir un accès à internet à partir de la tablette (de plus en plus courant). Ce système permettrait aussi d'envoyer sur le serveur (et donc de centraliser) les résultats et les statistiques de chaque tablette.
- Lors de la dernière présentation de l'application mobile à l'équipe d'INNOPHYT, celle-ci avait soumis l'idée d'une représentation graphique des campagnes, des parcelles et des pièges sur une carte (la

position étant obtenu via le capteur GPS). Cette fonctionnalité n'ayant pas été prévue dans le cahier de spécification écrit au début du projet, elle n'a pas été implémentée. Cependant, voici quelques pistes pour la réaliser.

Les coordonnées GPS sont déjà utilisées et enregistrées par l'application, il manque simplement l'affichage. Le hors-série de LINUX magazine/France de mai-juin 2012 explique comment faire. Voici la démarche en quelques mots :

Le chapitre : "Implémenter une Google Map" est consacré à la mise en place et à la gestion de la carte. Pour utiliser une Google Map, il faut une clé API. Pour cela, il vous faut le code MD5 de votre application et aller sur le site [Google Code](#). Une nouvelle clef est donnée par le site, sous réserve d'acceptation des termes d'utilisation. Cette clef est liée au compte Google qui en a fait la demande, ce qui peut poser des problèmes lors d'un développement collectif sur des ordinateurs différents. Dans tous les cas, ces deux clefs vont de pair, et en cas de problème, il faut les régénérer. L'article explique également comment intégrer cette carte à l'application et comment la contrôler. Nous ne rentrerons pas dans les détails car tout y est très bien détaillé.

Le second chapitre qui nous intéresse s'intitule : "Géolocalisation sur une Google Map". C'est le deuxième point sur lequel on va s'attarder car il y est expliqué comment ajouter des points sur la carte.

Une chose qui est passée sous silence dans ce magazine, c'est le dessin de zone sur la map. Pour cela, il faut que les zones soient contenues dans un fichier au format KML (Keyhole Markup Language). Ce langage est utilisé pour l'affichage de données dans tous les outils Google pour la localisation.

L'insertion se fait très simplement en utilisant la méthode addOverlay() de l'objet map.

Comme pour créer un overlay depuis un fichier KML, il faut utiliser GGeoXml : map.addOverlay(new GGeoXml('monfichier.kml'));

Conclusion

Ce projet collectif a été une expérience très enrichissante pour toute l'équipe. Il nous a permis de mener à bien un projet long et complet. Cela permet de se forger une première expérience de ce qui pourrait se passer en entreprise. Dans l'état actuel des choses, le projet n'est pas tout à fait terminé, principalement au niveau de l'application PC à laquelle il manque encore plusieurs fonctionnalités. Une base solide a malgré tout été posée, et le format de données XML offre en tous cas une approche plus simple et plus pratique pour l'édition des données que la solution existante. L'application mobile est quant à elle entièrement fonctionnelle. L'équipe INNOPHYT pourra dès maintenant proposer une phase de test en condition réelle de l'application tablette à ses collaborateurs pour obtenir des premiers retours.

Documentation utilisateur de l'application mobile

A.1 Introduction

Bienvenue dans l'application d'INNOPHYT pour la reconnaissance d'insectes nuisibles. Cette documentation vous guide pas à pas dans son utilisation.

A.2 Objectif de l'application

Cette application est utilisée par des biologistes de terrain pour déterminer si les insectes capturés sont nuisibles. En fonction des informations indiquées et renvoyées, INNOPHYT aura connaissance de ces insectes. La sauvegarde d'informations se fait au travers de projets définis dans le temps et dans l'espace. Ce qui permet de suivre l'évolution des insectes géographiquement et temporellement.

L'application est conçue pour être utilisée par plusieurs personnes. C'est pourquoi il est nécessaire de rentrer son nom avant toute utilisation. Chaque biologiste possède ses propres projets, ceux-ci sont nommés campagnes et sont découpés en parcelles et en pièges.

A.3 Créer et configurer un compte

Dès l'ouverture du logiciel, toutes les fonctionnalités ne sont pas disponibles. Vous devez vous créer un compte ou si c'est déjà fait, il faut vous connecter.

Pour cela, allez dans le menu à l'onglet "utilisateur" et appuyez dessus. Vous devriez avoir une vue comme celle-ci :

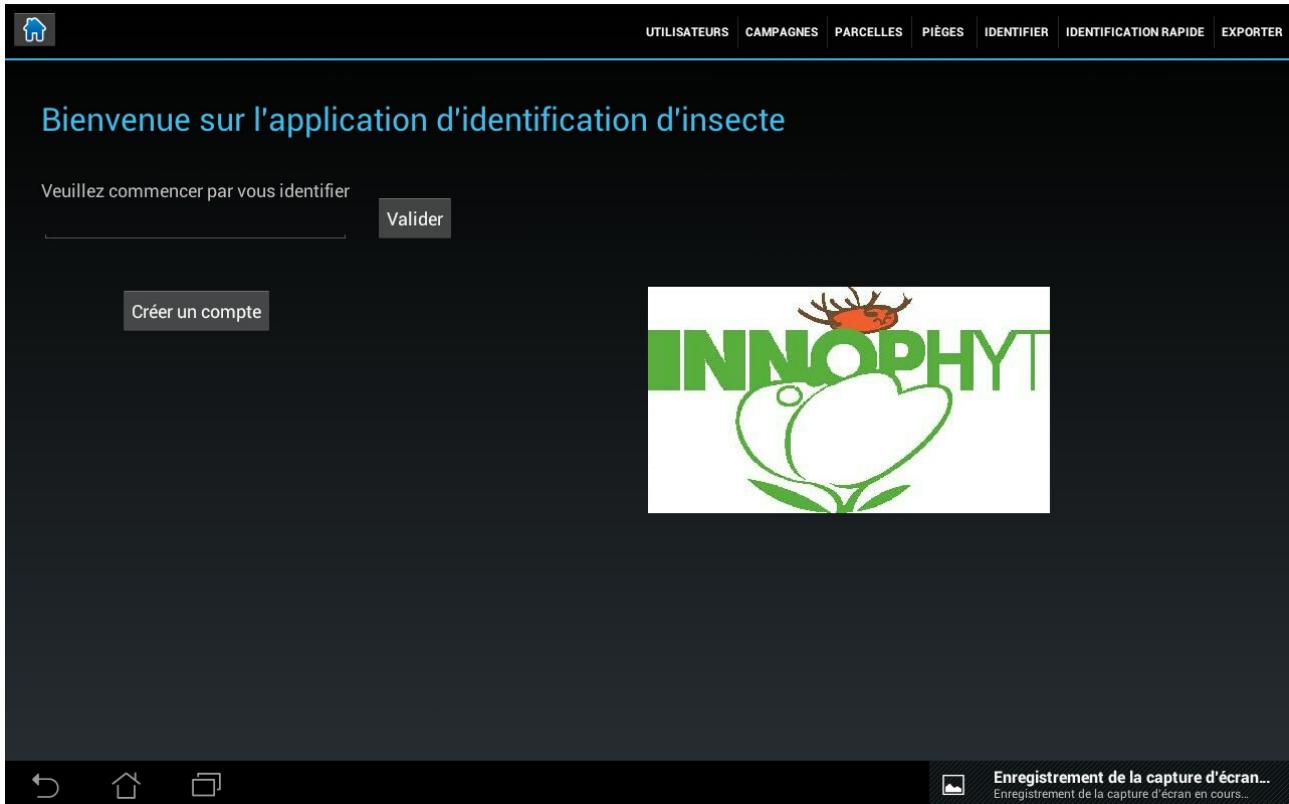


Figure A.1: Interface de connexion à l'application mobile

Dans le champ texte qui est juste en dessous de "Veuillez commencer par vous identifier" indiquez votre nom et appuyez sur "Créer un compte".

Si vous possédez déjà un compte sur ce terminal, rentrer votre nom et appuyez sur "Valider". Cette action vous mène directement sur la vue vous permettant de gérer vos campagnes. Au début, elle doit ressembler à ceci :

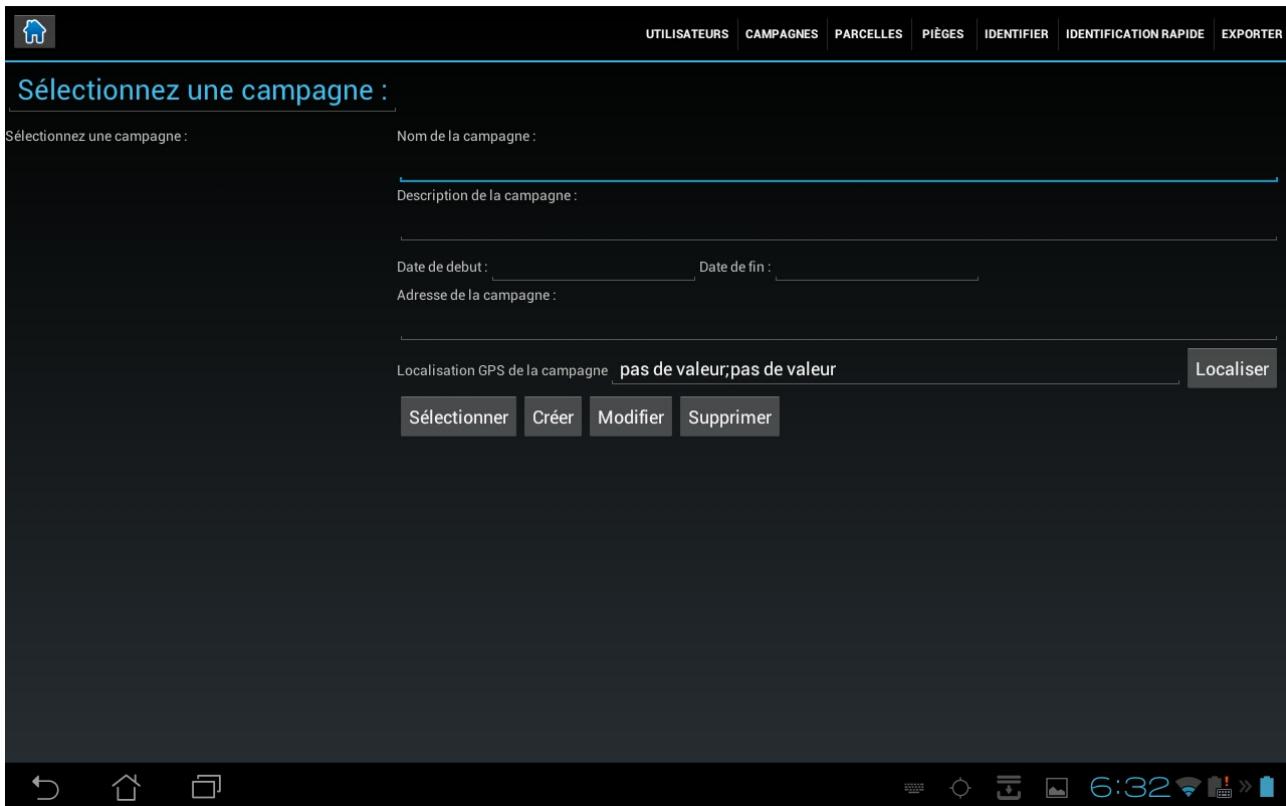


Figure A.2: Interface de sélection d'une campagne

Cette interface est découpée en deux parties distinctes. La colonne de gauche liste vos campagnes, au départ vous n'en avez pas, c'est normal, il faut que vous les créiez. Lorsque vous en posséderez plusieurs, vous pouvez en sélectionner une en appuyant sur son nom.

Dans la partie droite se trouvent toutes les informations relatives à la campagne en cours de création ou qui est sélectionnée. Ces informations sont les suivantes :

- Le nom
- La description
- La date de début
- La date de fin
- L'adresse
- La position PGS

Pour remplir les dates, soit, vous le faites à la main avec le clavier ou par l'interface en appuyant deux fois sur le champ de la date à écrire ; un calendrier apparaît.

Pour indiquer votre position avec le GPS, si tenté que cela définisse la position de votre campagne, appuyez sur le bouton "Localiser". Si un message d'erreur apparaît, c'est que la tablette ne connaît pas encore sa position, attendez quelques instants et recommencez.

Pour configurer convenablement une campagne, il est important d'être le plus précis possible. Cependant, seul le nom est obligatoire.

Pour la gestion des campagnes, les options suivantes sont disponibles :

- Créer : enregistre une nouvelle la campagne. Ne marche pas si une autre campagne porte le même nom.
- Modifier : enregistre les modifications portées à la campagne.
- Supprimer : supprime définitivement la campagne ainsi que les parcelles, les pièges et les insectes associés.
- Sélectionner : indique sur quelle campagne vous voulez travailler et vous permet de passer à la gestion des parcelles.

Il en va de même pour la configuration et la sélection des parcelles et des pièges. Attention, pour ces derniers, il subsiste une légère différence. En plus du nom, la date de fin est obligatoire. C'est ce qui détermine à quel moment les insectes dans le piège ont été récupérés.

A.4 L'identification

Une fois votre campagne, votre parcelle et votre piège sélectionnés, vous pouvez commencer l'identification. Si vous ne le faites pas, les résultats ne seront pas enregistrés.

Une série de questions vous sera posée ; pour chaque question vous aurez une liste de réponses sur la partie droite de l'écran. Chacune de ces réponses sera accompagnée d'images et/ou de textes pour vous aider dans votre choix. De plus, en dessous de la question posée, vous avez l'affichage en direct de la caméra arrière de votre tablette. Si vous avez besoin de comparer en particulier un insecte, à une image de votre caméra, il vous suffit de tapoter sur la zone d'affichage de la caméra. Cela aura pour effet de prendre une photo de l'insecte, ensuite dans le panel des réponses, appuyez longuement sur la photo avec laquelle vous souhaitez comparer l'image sauvegardée. Une fenêtre de comparaison s'ouvre alors, une fois votre analyse terminée, appuyez sur une des deux images pour fermer la fenêtre. Pour sélectionner une réponse, il vous suffit d'appuyer sur le texte de la réponse ou d'appuyer sur la case à cocher à côté du texte. Soit cela vous mène à d'autres questions, soit une petite fenêtre s'ouvre et vous propose un insecte. Dans ce deuxième cas, s'il correspond, vous pouvez enregistrer votre résultat en indiquant le nombre qu'il y en a dans votre piège.

Si vous vous êtes trompé dans votre choix de réponse, ce n'est pas grave une flèche en haut à gauche de l'application est disponible pour vous permettre de naviguer dans vos choix déjà effectués, un peu comme l'historique de votre navigateur internet.

Si vous vous retrouvez bloqué, c'est-à-dire que vous ne trouvez pas de réponse qui vous convienne, remplissez un rapport d'incident en appuyant sur le gros triangle orange en haut de l'interface. Décrivez votre problème, cela aidera l'équipe INNOPHYT à améliorer l'application.



Figure A.3: Interface d'identification d'un insecte

Sur cette capture d'écran (figure A.3, on observe le titre au-dessus de la zone verte qui correspond à la vue camera. Sur le côté se trouve la liste des réponses.

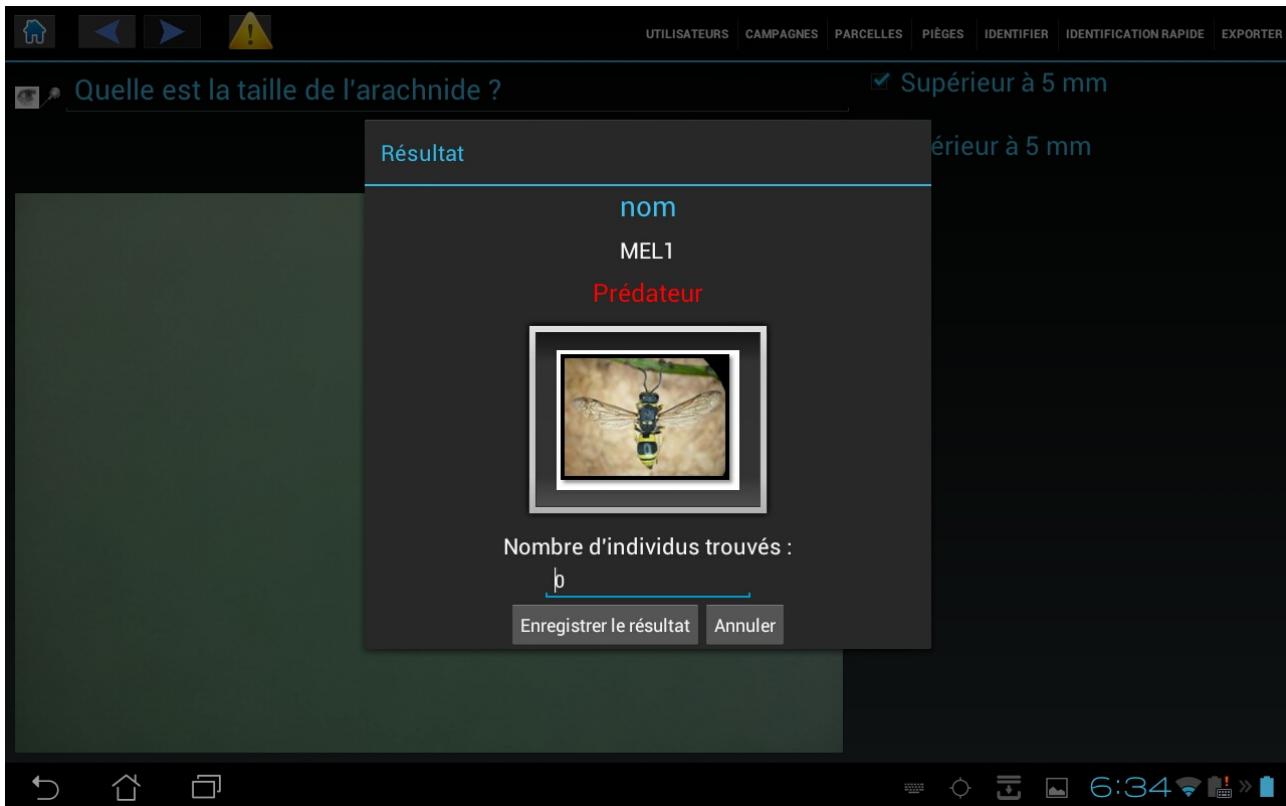


Figure A.4: Interface d'affichage du résultat

La vue de la figure A.4 est obtenue quand vous avez répondu à toutes les questions.

A.5 L'identification rapide

Une fois votre campagne, votre parcelle et votre piège sélectionnés, vous pouvez commencer l'identification. Si vous ne le faites pas, les résultats ne seront pas enregistrés.

Vous serez amené à utiliser cette partie si vous avez reconnu un insecte et que vous ne voulez pas avoir à répondre aux questions.

Sélectionnez dans le menu, l'entrée "Identification rapide", cela vous montre une mosaïque d'insectes (figure A.5). En appuyant sur l'image de l'insecte que vous recherchez, une nouvelle vue (figure A.6) apparait et vous demande le nombre que vous en avez dans votre piège.

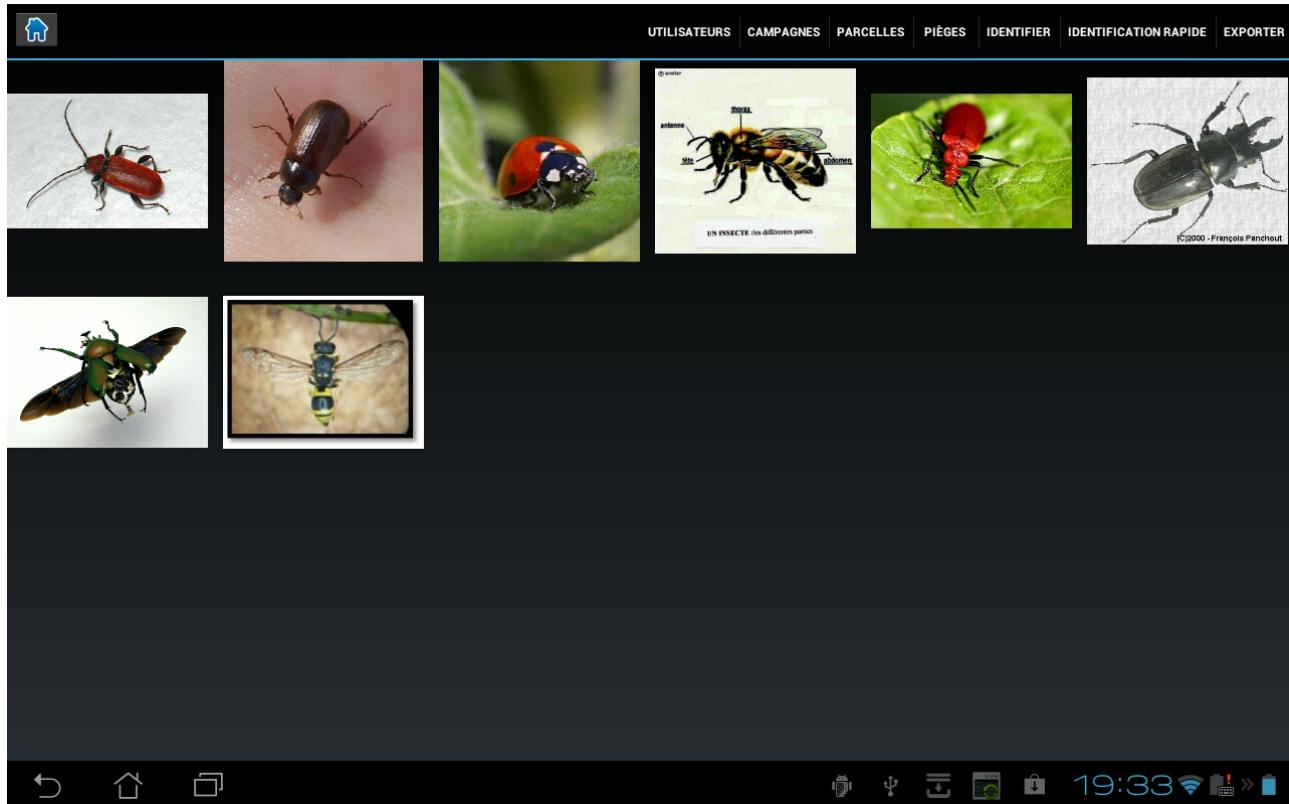


Figure A.5: Mosaïque des insectes pour l'identification rapide

A.6 L'exportation

L'exportation est la phase la plus importante, l'ensemble des données que vous avez saisi dans le terminal est enregistré dans un format texte utilisable par INNOPHYT.

Pour faire cet export, vous devez vous connecter en indiquant votre nom et appuyez sur l'entrée "Exporter" dans le menu. Un message apparaît vous indiquant où trouver ce fichier qui contient tout votre travail.

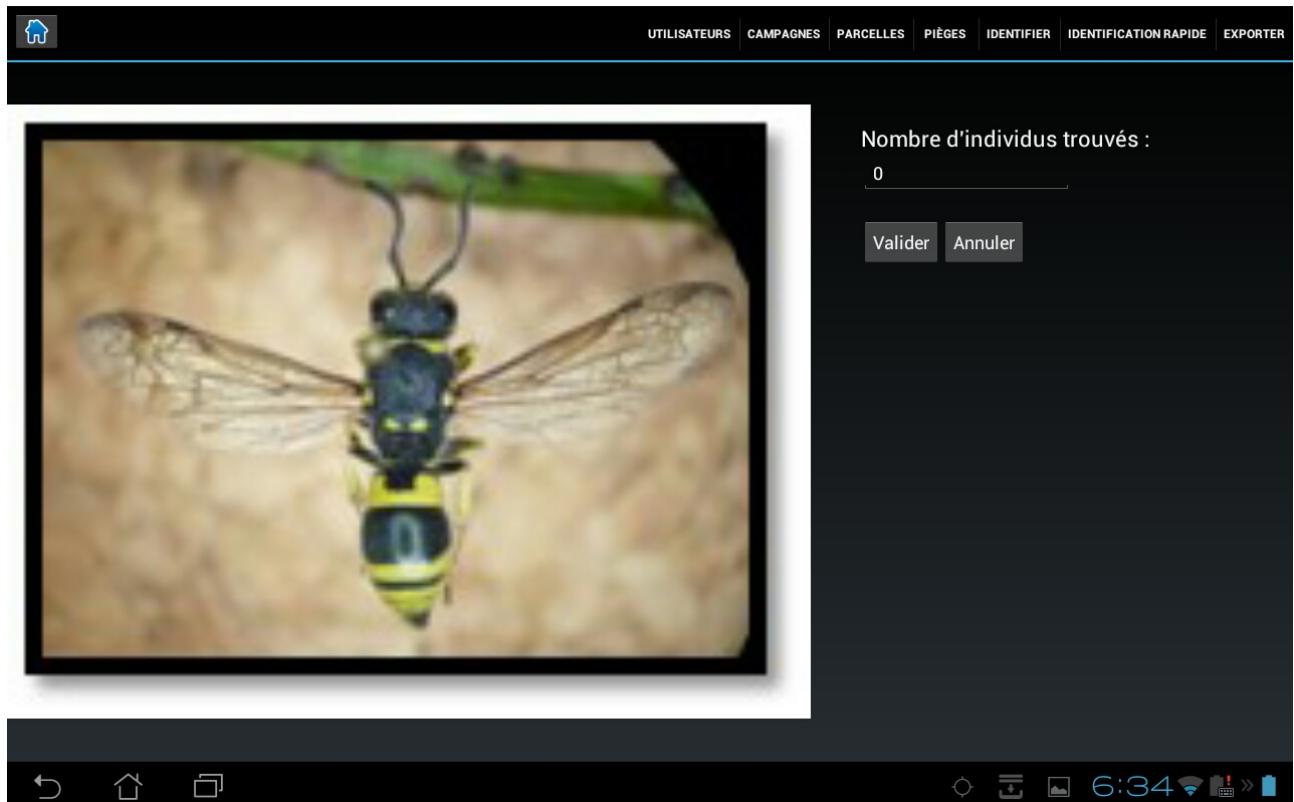


Figure A.6: Sauvegarde du résultat de l'identification rapide

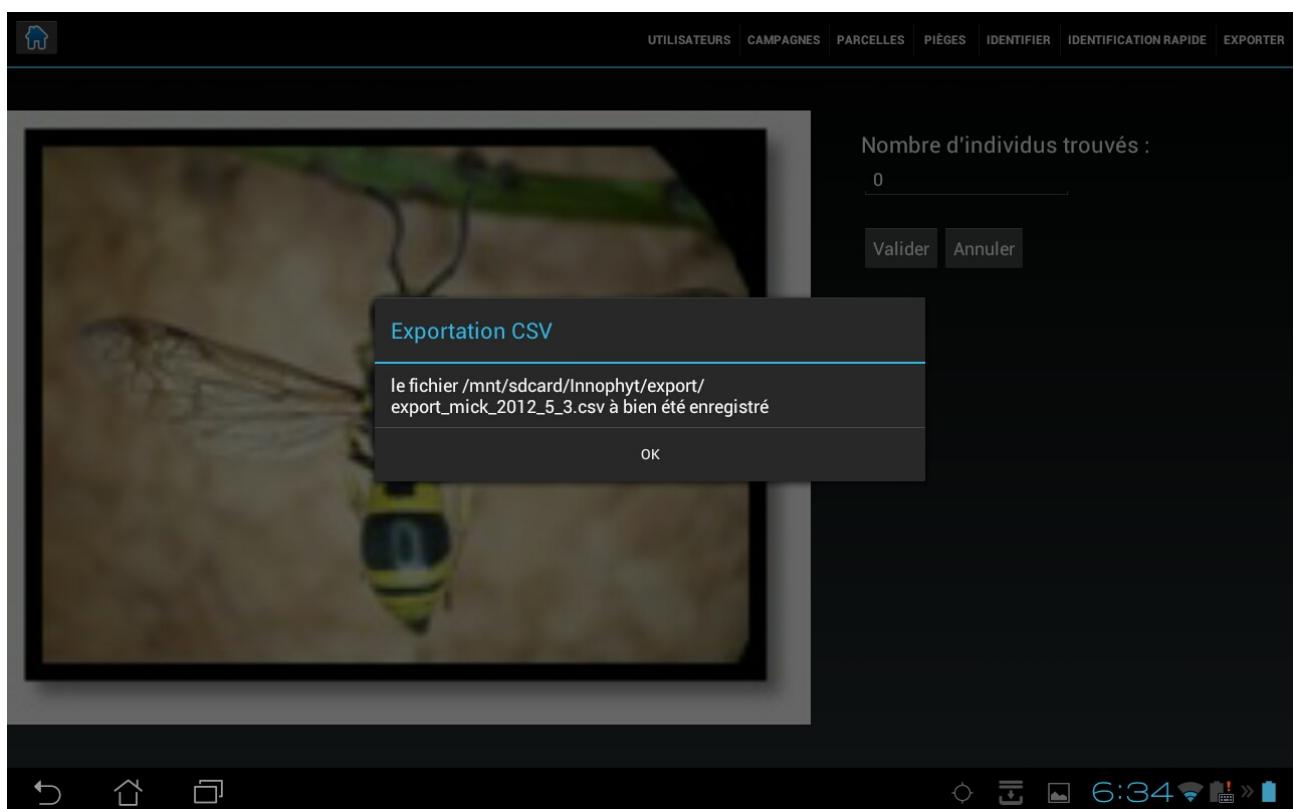


Figure A.7: Exportation des résultats au format CSV

Application d'aide à l'identification d'insectes nuisibles

Département Informatique
4ieme année
2011 - 2012

Rapport projet collectif

Résumé : L'objectif du projet était de réaliser deux applications pour l'équipe INNOPHYT de l'université de Tours. La première est une application mobile destinée à permettre l'identification d'un insecte grâce à une succession de questions et la deuxième est une application PC permettant de modifier la base de données contenant les questions, les réponses et les médias. Ce rapport est la synthèse de tout le projet, des phases de modélisation aux phases de tests en passant par les phases de développement et les parties gestion de projet. Ce rapport est un guide technique, qui pourra servir à un développeur désireux de reprendre et de comprendre les applications existantes, mais aussi un guide utilisateur, qui permettra à l'usager de trouver les réponses à ses questions à propos du fonctionnement des logiciels.

Mots clefs : qt, java, android, insectes, xml, innophyt, morpho espèce

Abstract: The aim goal of this project was to realize two applications for the team INNOPHYT of the university of Tours. The first one is a mobile application intended to allow the identification of an insect thanks to a succession of questions and the second is an pc application allowing to modify the database containing the questions, the answers and the media. This report is the synthesis of the whole project, from the phases of modelling to the phases of tests through the phases of development and some project management. This report is a technical guide, who can be used by a developer willing to learn and understand the existing applications, but it is also a user guide, which allows a user to find the answers to his questions about the softwares operation.

Keywords: qt, java, android, insects, xml, innophyt, species

Encadrants

Gilles VENTURINI

gilles.venturini@univ-tours.fr

Université François-Rabelais, Tours

Client

Ingrid ARNAULT

ingrid.arnault@univ-tours.fr

Damien MUNIER

munier.damien@aliceadsl.fr

Alexandre DEPOILLY

alexandre.depoilly@etu.univ-tours.fr

Équipe CETU INNOPHYT

Étudiants

Matthieu ANCERET

matthieu.anceret@etu.univ-tours.fr

Jérôme HEISSLER

jerome.heissler@etu.univ-tours.fr

Julien TERUEL

julien.teruel@etu.univ-tours.fr

Martin DEMEULEMEESTER

martin.demeulemeester@etu.univ-tours.fr

Mickael PURET

mickael.puret@etu.univ-tours.fr

Simon FAUSSIER

simon.faussier@etu.univ-tours.fr

Zheng ZHANG

zheng.zhang@etu.univ-tours.fr

Zhengyi LIU

zhengyi.liu@etu.univ-tours.fr

DI4 2011 - 2012