DOCUMENTATION TECHNIQUE

Sprint 2 : Affichage des visiteurs

DIANA Victor
MONSERRAT Laetitia
STEFANELLI Romain
TALTAVULL Gaëtan

Sommaire

Affichage des visiteurs

Début du codage de l'application en correspondance avec le modèle MVC et correction des erreurs.

Tâches principales

Organisation de l'application en M.V.C.

MVC est un design pattern, une bonne pratique de programmation qui recommande de découper son code en trois parties qui gèrent les éléments suivants :

Modèle : stockage des données

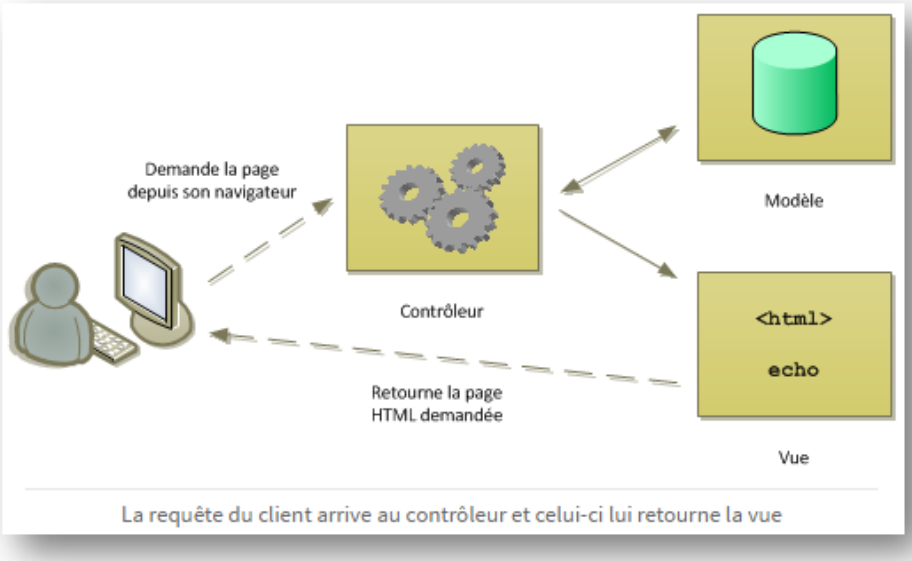
Vue : affichage de la page

Contrôleur : logique, calculs et décisions

Utiliser l'architecture MVC est recommandé, car cela permet d'avoir un code plus facile à maintenir et à faire évoluer.

L'architecture MVC permet de séparer l'affichage des informations, les actions de l'utilisateur et l'accès aux données. MVC signifie Modèle-Vue-Contrôleur. C'est un modèle qui a été conçu au départ pour des applications dites « client lourd » (dont la majorité des données sont traitées sur le poste client. Par exemple : un traitement de texte comme Word). MVC était tellement puissant pour ces applications, qu'il a été massivement adopté comme modèle pour la création d'applications web dites « client léger ».

Dans notre application, le dossier Modèle est divisé en trois parties (jdbc, métier, dao) qui remplissent chacun un des trois rôles du Modèle dans l'ordre : accès à la base, fonctions métier, gestion des erreurs.



La requête du client arrive au contrôleur et celui-ci lui retourne la vue

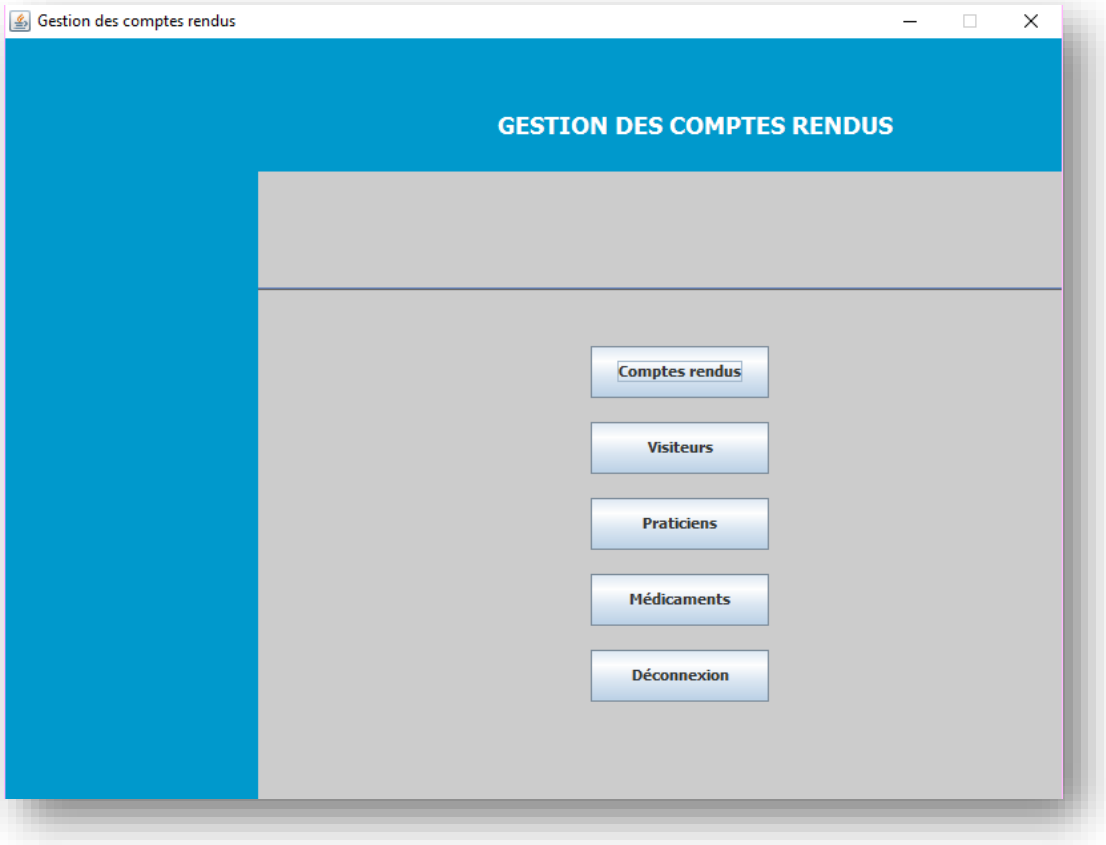
Modèle : cette partie gère les données du site. Son rôle est d'aller récupérer les informations brutes dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le Contrôleur. On y trouve donc les requêtes SQL.

Vue : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher.

Contrôleur : cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au Modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la Vue. Le Contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).

Spécification, Codage et Tests de la fonctionnalité

Depuis la Vue menu, seul le bouton déconnexion est opérationnel.



Lorsque l'on clique sur les autres boutons, il ne se passe rien.

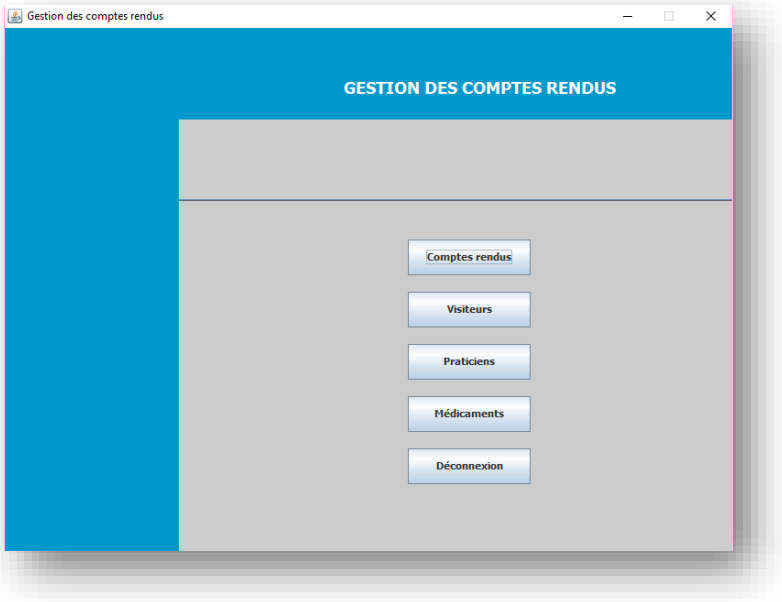
Nous voulons faire en sorte d'afficher les visiteurs.

Afin d'intégrer la nouvelle fonctionnalité « Affichage des visiteurs » nous avons tout d'abord codé sa Vue à l'aide de Swing :

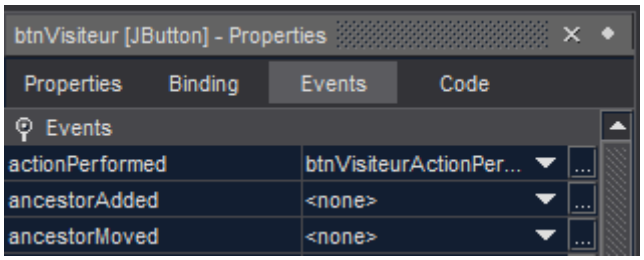
```java
/**
 * Creates new form VueVisiteurs
 */
public VueVisiteur(CtrlAbstrait ctrl)
{
    super(ctrl);
    initComponents();
    modeleJComboBoxVisiteur = new DefaultComboBoxModel();
    modeleJComboBoxLabo = new DefaultComboBoxModel();
    modeleJComboBoxSecteur = new DefaultComboBoxModel();
    jComboBoxVisiteur.setModel(modeleJComboBoxVisiteur);
    jComboBoxLabo.setModel(modeleJComboBoxLabo);
    jComboBoxSecteur.setModel(modeleJComboBoxSecteur);

                    // affiche un titre dans la barre
this.setTitle("Praticiens");
// modifie la taille de la fenêtre
this.setSize(800, 700);
//positionne la fenêtre au centre
this.setBackground(Color.blue);
this.setLocationRelativeTo(null);
// permet de fermer l'application en cliquant sur la croix rouge
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// Affiche la fenêtre
this.setVisible(true);
// false = empeche de modifier la taille de la fenetre =! true
this.setResizable(false);
// true = bloque la fenêtre au premier plan
this.setAlwaysOnTop(true);
}
```

Extrait de code de la Vue Visiteur

Depuis la vue menu, il a fallu rendre le bouton Visiteur opérationnel



Rendu de la vue menu



Ajout d'un évènement lors du clic du bouton Visiteur dans la VueMenu

```
private void btnVisiteurActionPerformed(java.awt.event.ActionEvent evt) {
    ((CtrlMenu)controleur).menuVisiteur();
}
```

Appel du controleur menu afin d'utiliser la méthode menuVisiteur() dans la VueMenu

```
public void menuVisiteur() {
    this.getCtrlPrincipal().action(EnumAction.MENU_VISITEUR);
}
```

Appel de l'action MENU_VISITEUR du controleur principal dans le controleur menu

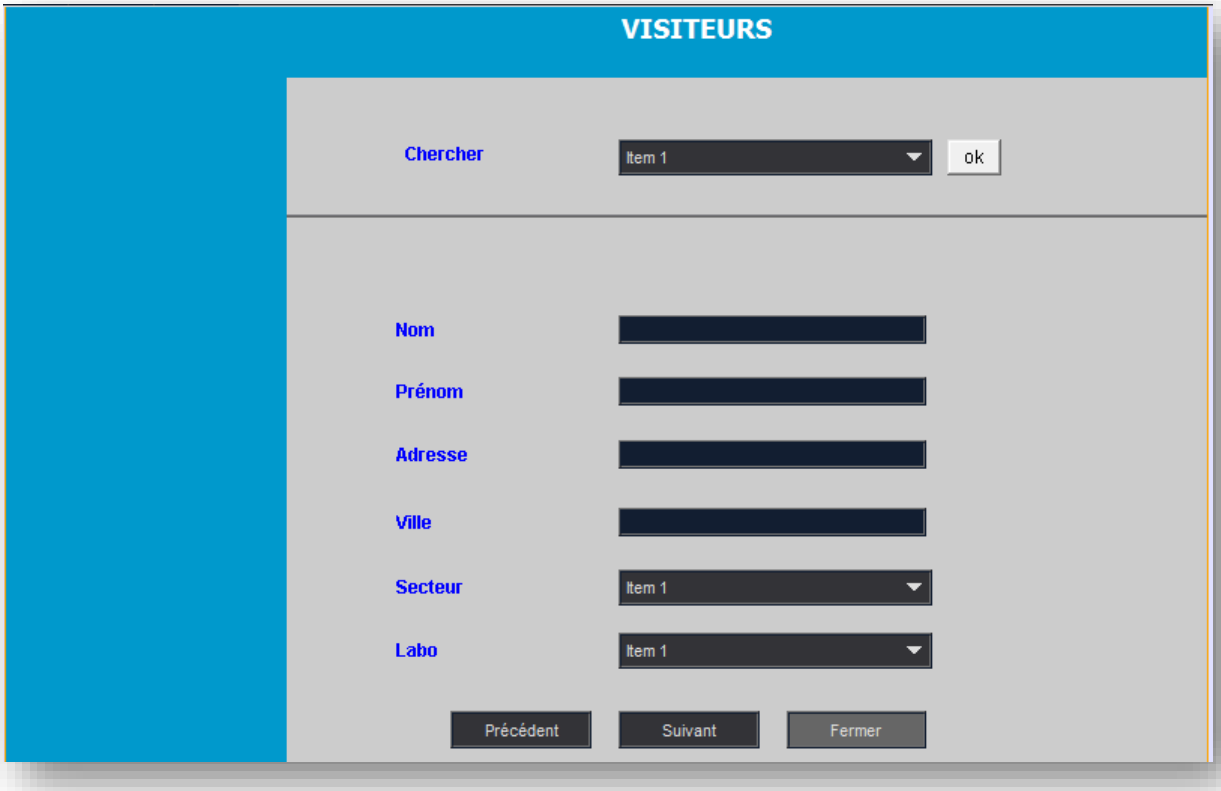
```
public void action(EnumAction action) {
    switch (action) {
        case MENU_VISITEUR: // activation de vueVisiteur depuis vueMenu
            menuVisiteur();
            break;
```

Appel de la méthode menuVisiteur() du controleur principal

```
/**
 * Transition vueMenu / vueVisiteur
 */
private void menuVisiteur() {
    if (ctrlVisiteur == null) {
        ctrlVisiteur = new CtrlVisiteur(this);
    } else {
        // si la le contrôleur et sa vue existent déjà
        // il faut rafraîchir le contenu à partir de la base de données
        ctrlVisiteur.actualiser();
    }
    // vueVisiteur est une fenêtre modale :
    // -> vueMenu reste visible, mais n'est pas active
    ctrlMenu.getVue().setEnabled(false);
    ctrlVisiteur.getVue().setVisible(true);
}
```

Méthode menuVisiteur() dans le controleur principal

Le bouton visiteur du menu envoie maintenant sur la vue Visiteur



Vue Visiteurs

Ensuite pour que les champs renseignent les informations des visiteurs, il faut compléter le controleur Visiteur

```
/**
 * visiteurSelectionner renseigner le modèle des composants
 * JTextField (TxtNom, TxtPrenom, TxtAdrs, TxtVille, TxtCp),
 * JComboBoxSecteur, JComboBoxLabo
 *
 */
public void visiteurSelectionner (){
    Visiteur visiteurSelect = (Visiteur) getVue().getjComboBoxVisiteur().getSelectedItem();
    if (visiteurSelect != null){

        getVue().getTxtNom().setText(visiteurSelect.getNom());
        getVue().getTxtPrenom().setText(visiteurSelect.getPrenom());
        getVue().getTxtAdrs().setText(visiteurSelect.getAdresse());
        getVue().getTxtVille().setText(visiteurSelect.getVille());

        getVue().getModeleJComboBoxSecteur().setSelectedItem(visiteurSelect.getSecteur());
        getVue().getModeleJComboBoxLabo().setSelectedItem(visiteurSelect.getLabo());
    }
}
```

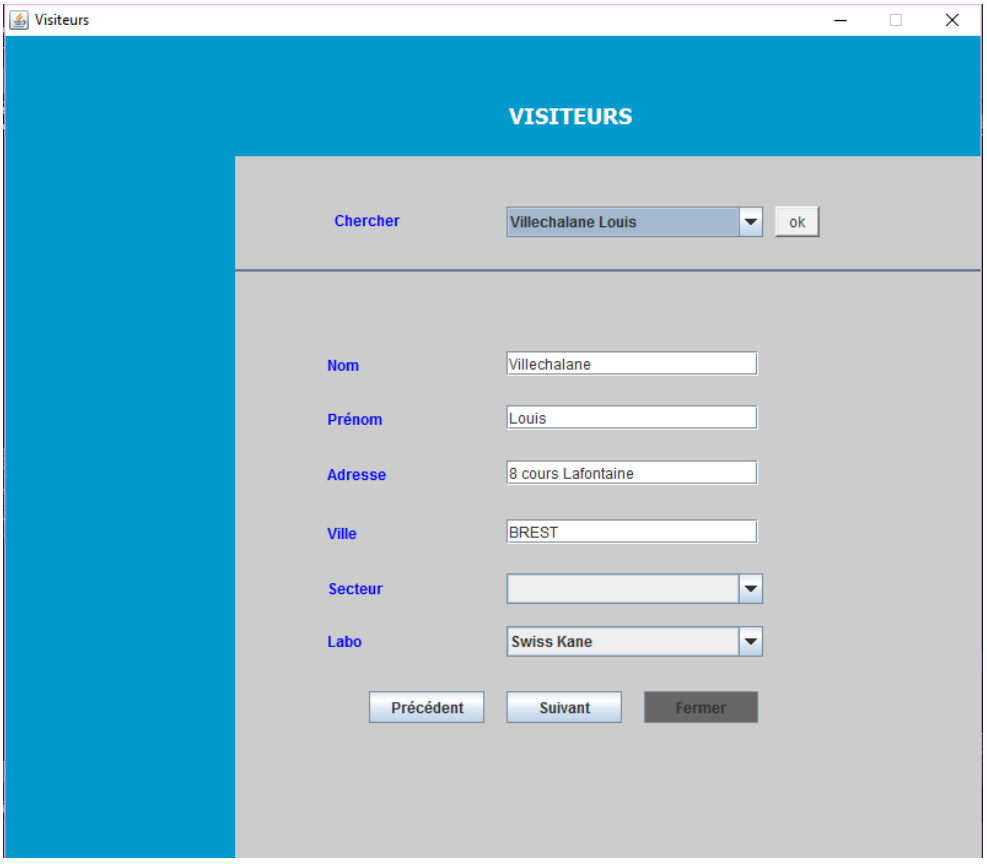
Extrait du code controleur visiteur

Puis, il faut activer le bouton fermer de la même manière que l'on a activé le bouton visiteur dans la vue Menu

```java
/**
 * visiteurFermer réaction au clic sur le bouton Fermer de la vueVisiteur
 * Le contrôle est rendu au contrôleur frontal
 */
public void visiteurFermer() {
    this.getCtrlPrincipal().action(EnumAction.VISITEUR_FERMER);
}
/**Bouton suivant. rajoute +1 a chaque utilisateur
 *
 */
public void visiteurSuivant(){
    int index = getVue().getjComboBoxVisiteur().getSelectedIndex()+1;
    if(index== getVue().getjComboBoxVisiteur().getItemCount())index=0;
    getVue().getjComboBoxVisiteur().setSelectedIndex(index);
}

/**
 Bouton précédent enleve -1 a chaque utilisateur
 */
public void visiteurPrecedent(){
    int index = getVue().getjComboBoxVisiteur().getSelectedIndex()-1;
    if(index== -1) index=getVue() .getjComboBoxVisiteur() .getItemCount() -1;
    getVue().getjComboBoxVisiteur().setSelectedIndex(index);
}
```

Extrait des méthodes des boutons de la vue Visiteur dans le controleur Visiteur

Voilà, notre fonctionnalité « Afficher les visiteurs » est opérationnelle



Vue Visiteurs

Lorsque l'on clique sur le bouton précédent ou suivant on change de visiteur.
Lorsque l'on clique sur le bouton fermer on ferme cette vue et on retourne sur la vue menu.

Livrables Sprint 2

Livrables faisant parti du second Sprint :

- Le diagramme de classe UML métier
- Daily Scrum
- Le diagramme de Gantt prévisionnel (celui du Sprint 1)
- Le diagramme de Gantt actualisé
- La répartition des tâches et des fonctions
- Les rapports de tests
- Documentation Technique Sprint 1 (ce document même)

Le diagramme de classe UML métier

Le diagramme de classe permet d'avoir une bonne vue d'ensemble sur le fonctionnement de l'application. Ici on peut observer les classes du paquetage metier.

Daily Scrum

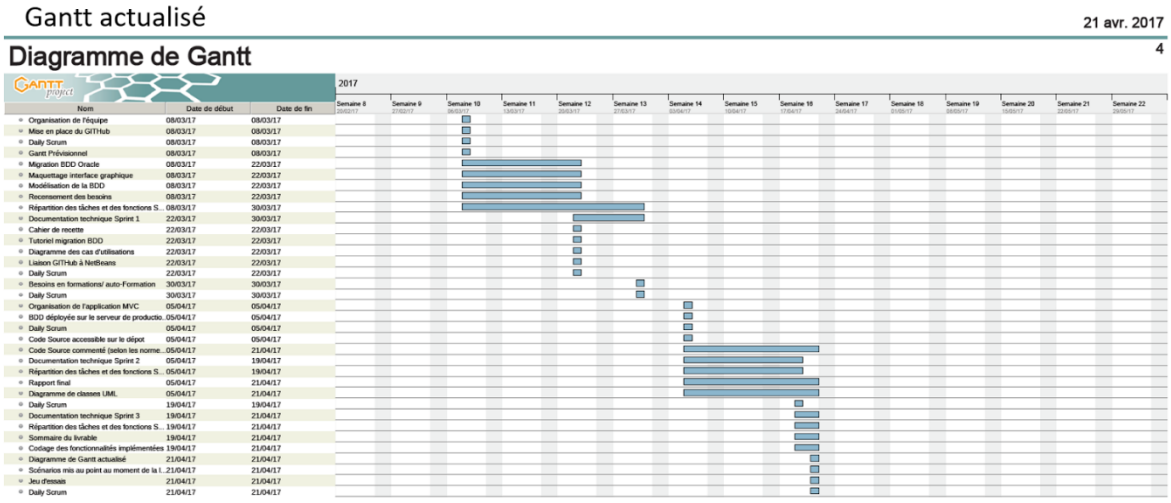
Le Daily Scrum (mêlée quotidienne) est une réunion de planification qui permet aux développeurs de faire un point de coordination sur les tâches en cours et sur les difficultés rencontrées.
Nous faisons un Daily Scrum tous les jours où nous travaillons sur le projet. Ce qui permet la synchronisation de l'équipe, l'évaluation de l'avancement vers l'objectif de l'itération, la collecte d'information nécessaire à l'auto-organisation.

Nous résumons ce qui a été fait, ce qui est en cours de réalisation et ce qui reste à faire.

Les diagrammes de Gantt

Le diagramme de Gantt permet de planifier de façon optimale ainsi que de communiquer sur le planning établi et les choix qu'il impose. Le diagramme permet de déterminer les dates de réalisation d'un projet, d'identifier les marges existantes sur certaines tâches ainsi que de visualiser d'un seul coup d'œil le retard ou l'avancement des travaux.

Nous avons actualisé notre premier diagramme de Gantt :



Répartitions des taches et des fonctions

Documents résumant la répartition des tâches et fonctions du projet.

Cahier de recette

Le cahier de recette contient la liste exhaustive de tous les tests pratiqués sur l'application existante. Il résume les erreurs et les modifications qu'il va falloir apporter à l'application.
Différents scénarios ont été pris en compte, en effet se mettre dans la peau d'un utilisateur lambda aide à parfaire une application. Quels chemins va-t-il prendre ? Quelles subtilités va-t-il essayer ?

Il faut tout envisager et essayer toutes les possibilités.

Grâce à cette approche, de nombreux bugs sont identifiés et donc corrigés au fur et à mesure. Par exemple un utilisateur ayant des droits de base ne doit pas pouvoir modifier des paramètres autres que les siens.

De nombreux dysfonctionnements provenant de l'application existante ont été découvert grâce à ce procédé. Afin de satisfaire pleinement l'utilisateur. L'application doit être fiable.