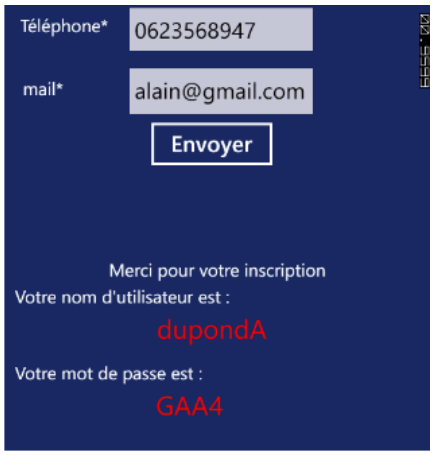
Projet Covoiturage version Windows Phone TP 4

Description du thème

Propriétés	Description
Intitulé long	Quatrième TP d'une série permettant la découverte d'une application mobile sous les systèmes d'exploitation Windows Phone 7, 7.5 ou 8
Formation concernée	BTS Services informatiques aux organisations
Matière	PPE, SLAM 4
Présentation	Ces TP proposent de développer une application mobile à différentes itérations du cycle de développement
Notions	Savoirs • D4.1 - Conception et réalisation d'une solution applicative • D4.2 - Maintenance d'une solution applicative Savoir-faire • Programmer un composant logiciel • Exploiter une bibliothèque de composants • Adapter un composant logiciel • Valider et documenter un composant logiciel • Programmer au sein d'un framework
Pré-requis	Développement objet, C #, Visual-Studio
Outils	Un environnement de développement pour les applications mobiles Windows Phone. Visual Studio 2010 express pour application Windows Phone au minimum ou Visual Studio 2010 et le kit de développement Windows Phone.
Mots-clés	Application mobile, C#
Durée	2 h
Auteur(es)	Patrice Grand
Version	v 1.0
Date de publication	Mars 2015

Énoncé

La suite du développement doit permettre de créer le formulaire d'inscription qui est proposé à partir du formulaire de login.
Le formulaire attendu devra ressembler à ce qui suit ; à la validation, la zone inférieure devra faire apparaitre les valeurs de *user* et *mot de passe* :



Le nom de *user* est constitué du nom en minuscule et de la première lettre du prénom en majuscule. Le mot de passe est aléatoire.

Des contrôles de validité de saisie portent sur les champs nom (non vide), téléphone (10 chiffres) et mail (adresse sémantiquement valide). Un ou des messages d'erreurs apparaissent si une ou des erreurs de saisie se produisent :

Avant de vous lancer dans ce travail, votre collègue vous fait part de son travail sur ce projet :

<< J'ai commencé à développer ce formulaire, j'ai traité ce qui concerne le champ de saisie du nom, j'ai effectué la validation de ce champ et écrit dans la partie modèle le code nécessaire, jette un coup d'œil et n'hésite pas à me poser des questions si des choses te paraissent obscures >>

Vous prenez connaissance du code ; débutant dans cette technologie, vous ne manquez pas de lui poser de nombreuses questions.

<<

- *Vous : À vrai dire, je ne comprends pas grand chose au mécanisme de validation…*
- *Votre collègue : Oui, ça ne m'étonne pas ; c'est un peu particulier. Windows Phone propose différentes techniques de validations de saisie. Celle-ci est celle qui est supportée par Windows Phone 7 et Windows Phone 8.*
  L'idée c'est de binder chaque champ de saisie à un objet de la classe chauffeur ; ainsi à chaque saisie (dans l'interface), le Setter de la propriété concernée va être exécuté et…on va gérer les erreurs dans le modèle.
- *Vous : Et comment fait-on ?*

- *Votre collègue : À l'ouverture du formulaire pour l'inscription, on renseigne la propriété DataContext qui va pointer sur un chauffeur vide (comme pour le formulaire sur une offre) :*

```
public modele.Chauffeur unChauffeur;
public inscription()
{
    InitializeComponent();
    unChauffeur = new modele.Chauffeur();
    this.LayoutRoot.DataContext = unChauffeur;
}
```

LayoutRoot est le conteneur principal (un Grid). C'est à lui que sera attaché le contexte de binding, donc un objet Chauffeur. Chaque champ sera bindé à une propriété de l'objet (dans une propriété xaml).

```
<TextBox Height="71" HorizontalAlignment="Left" Margin="117,-5,0,0"
    Name="txtNom" VerticalAlignment="Top" Width="231"
    Text="{Binding  nom, NotifyOnValidationError=True,
    Mode = TwoWay, ValidatesOnExceptions =True }"/>
```

NotifyOnValidationError = True et ValidatesOnException =True vont permettre d'associer un événement à une erreur signalée dans le Setter de la propriété. Le mode TwoWay indique que le binding est dans les deux sens.
La suite doit être faite dans le modèle avec une levée d'exception en cas d'erreur :

```
public string nom {
    get { return _nom; }
    set
    {
            if (!String.IsNullOrEmpty(value))
                _nom = value;
            else
                throw new Exception("Le champ nom ne doit pas être vide !!");

    }
}
```

Pour terminer, le panel qui contient les champs de saisie a une propriété (qui est un événement) qui va lui permettre de gérer les erreurs :

```
BindingValidationError="StackPanel_BindingValidationError" >
```

Cette propriété pointe sur un événement dans lequel on va enregistrer les évolutions des erreurs :

```
private void StackPanel_BindingValidationError(object sender, ValidationErrorEventArgs e)
{
    if (e.Action == ValidationErrorEventAction.Added)
    {
        errors.Add(e.Error);
    }
    else
        if (e.Action == ValidationErrorEventAction.Removed)
        {
            errors.Remove(e.Error);
        }
    this.listBoxErreur.ItemsSource = this.errors;
}
```

- Vous : C'est quoi errors ?
- Votre collègue : C'est une liste un peu particulière, de type ObservableCollection ; mais qui fonctionne comme une liste normale ; elle a été déclarée comme champ privé dans la classe.

```
private ObservableCollection<ValidationError> errors = new ObservableCollection<ValidationError>();
```

- Vous : Que signifie cette ligne :

```
private void btnEnvoyer_Click(object sender, RoutedEventArgs e)
    {
        this.txtNom.GetBindingExpression(TextBox.TextProperty).UpdateSource();
```

- Votre collègue : Si l'on valide le formulaire sans rien saisir, le gestionnaire d'erreur ne réagit pas, cette ligne force la mise à jour du champ.
- Vous : Ok …
- Votre collègue : Pour gérer l'affichage du nom de user et du mot de passe, je te conseille d'utiliser également le binding.
- Vous : Ok…

Travail à faire
Terminer le formulaire en tenant compte des remarques de votre collègue. Temps estimé : 2 heures