Laboratoire: VUES et CONTRÔLEUR GÉNÉRÉS POUR Gestion d’UTILISATEURS MAISON

# Objectif(s)

* Views et Controllers auto-générés
* Validations complexes CRUD Révision:
  + ViewsModels vs Models
  + Views Spéciales
  + Traduction pour les validateurs

# Créer la branche de la fonctionnalité

Clonage du repository

1. Dans Gitkraken, clonez le repository
   * Soit via URL
   * Soit via GitHub.com
2. Ouvrez le repository (OK)

Création d’une branche pour la fonctionnalité

1. Créez une branche appelée **FCT \_Vues\_Users**
2. Assurez-vous d'être dans la branche que vous avez créée dans LOCAL

# VUES ET CONTRÔLEUR D’UTILISATEUR

## Générez les vues pour User

1. Allez dans le répertoire **/Controllers** :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

1. Il faut choisir l’option MVC avec vue qui utilise Entity Framework

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

* Générez pour le modèle **User**

Note : si l’auto-génération vous donne des erreurs, utilisez la ligne de commande suivante dans PS (fenêtre console PwerShell), à **partir du dossier du PROJET**, PAS de la SOLUTION.

Rappel : Majuscule-boutonDroit de la souris dans le dossier du projet, Ouvrir une fenêtre PowerShell ici.

dotnet aspnet-codegenerator controller -m PresseMots\_Web.Models.User -dc PresseMots\_Web.Models.Data.PresseMotsDbContext -scripts -udl -outDir Controllers -name UsersController -async -actions

* Lancez l’application et testez les vues générées avec le chemin d’accès **/Users**
* Bon, ce n’est vraiment pas super comme gestion d’usagers, alors vous avez pas mal de travail, mais vous devriez déjà pouvoir manipulez des **Users**
* Pour commencer, ajoutez un[EmailAddress] au champ Email de User, question d’avoir un minimum de validation!

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés:
   * **Summary** FCT ajout d’un contrôleur User avec ses vues
   * **Description**: Création du contrôleur UserController et des vues Index, Delete, Create, Details et Edit
5. Cliquez sur **Commit**

# MODIFICATION DES VUES D’UTILISATEUR

## Modifiez LES vue USER/index et USER/DELETE

1. Modifiez la vue **User/Index** pour ne pas afficher le **password**, mais afficher **le nombre de stories** écrite par chaque **User.**
2. Modifiez la vue **User/Delete** pour ne pas afficher le **password**

## Modifiez LA vue USER/CREATE

1. On va utiliser cette librairie pour tester la complexité du mot de passe :

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

1. On teste la force d’un mot de passe comme ceci:

var result = Zxcvbn.Core.EvaluatePassword(user.Password);

if (result.Score < 3)

ModelState.AddModelError("Password", "Password is too weak.");

1. Intégrez cette logique à votre contrôleur pour vérifier les mots de passe lorsque vous créer un nouveau User.
2. Créer un **UserCreateVM** qui va contenir les champs nécessaires pour créer un User en plus d’un champ **ConfirmPassword**.

//Mettre les autres champs nécessaire à la création du User

[DataType(DataType.Password)]

[Required(ErrorMessage = "Password is Required")]

public string Password

{

get;

set;

}

[DataType(DataType.Password)]

[Required(ErrorMessage = " Confirm Password is Required")]

[Compare("Password", ErrorMessage = "Password and Confirm Password do not match")]

public string ConfirmPassword

{

get;

set;

}

Note : L’annotation **Compare** permet de vérifier si deux champs sont égaux!

1. Modifiez votre vue pour utiliser **UserCreateVM** et affichez les nouveaux champs.
2. Vous devriez maintenant pouvoir créer un **User** seulement si le **Password** est assez complexe et si **ConfirmPassword** contient la même *string*.

## Modifiez LA vue USER/Edit

1. Utilisez la même librairie de vérification de mot de passe que pour le create.
2. Ajoutez un ViewModel **UserEditVM**. Cette-fois, on veut non seulement deux copies du mot de passe (**ConfirmPassword**), mais également l’entrée du mot de passe précédent **OldPassword**. Vous pouvez hériter de **UserCreateVM** pour créer **UserEditVM**.
3. Modifiez votre vue pour utiliser **UserEditVM** et affichez les nouveaux champs.
4. Assurez-vous que vous pouvez modifier un **User** seulement si vous entrez les 3 passwords correctement (**OldPassword**, **Password** et **ConfirmPassword**)

## Modifiez LA vue USER/DETAILS

1. Lorsqu’on clique sur le **details** d’un **User**, on veut voir : ses propres **stories**, les **stories** qu’il a « liked » et les **stories** qu’il a « shared ».
2. Ce n’est pas possible d’ajouter des likes et des shares dans l’application présentement, mais l’utilisateur Charlie a déjà mis un like à la première histoire et a fait un share de la deuxième. (Dans le *seed*)
3. Ajoutez un ViewModel **UserDetailsVM** pour faire ça.

public class UserDetailsVM

{

public string Name { get; set; }

public List<Story> AuthoredStories { get; set; }

public List<Story> LikedStories { get; set; }

public List<Story> SharedStories { get; set; }

}

1. Ajustez le code généré dans le contrôleur (Details) pour utiliser le ViewModel.
2. Inspirez vous de l’exemple 2 de S05.2\_Demo pour utiliser JQuery pour afficher/masquer chacune des sections de stories lorsque vous cliquez sur le titre (« Authored Stories », par exemple).

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés:
   1. **Summary** FCT modification des Vues de User
   2. **Description**: Cacher les mots de passes et permettre de bien valider les mots de passe à l’édition et à la création. Affichage des stories dans la vue User/Details.
5. Cliquez sur **Commit**

# Traduction et sUPPORT des décimales

## Faites la traduction de vos nouvelles pages

1. Ajoutez la configuration nécessaire pour faire la traduction et changer de langue. (Référez-vous aux laboratoires précédents)
2. Traduisez toutes les vues de **User** que vous avez créées (Edit, Create, Delete, Index et Details).

## Decimal et validation

1. Ajoutez un champ avec décimal pour le **User**

public float SocialCredit { get; set; }

1. Vérifier la création d’un User quand votre site est en français avec un social credit qui contient une virgule ou un point. (Ça devrait vous donner une erreur dans les deux cas!)
2. Ajoutez les librairies qui sont dans le dossier librairies-globalize.
3. Remplacer \_ValidationScriptsPartial par celui fourni.
4. Testez que ça fonctionne correctement maintenant

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés:
   1. **Summary** Traduction des pages de User et support des décimales
   2. **Description**: Traduction des textes des vues de User et ajout du support de la validation des décimales en français
5. Cliquez sur **Commit**

## Commentaires et validation (Commit) des changements dans le code

1. Une fois que tout est terminé, n’oubliez pas de faire un push.
2. Merge la branche **FCT \_Vues\_Users** dans la branche **Main** et faites un **dernier push.**

**Fin du laboratoire!**