Evaluation du module ASP.NET et ADO.NET

Partie 2 : Exercices

Consignes de l'examen :

Chaque exercice est indépendant mais devra être réalisé dans un unique projet ASP.NET MVC (.NET Core ou .NET Framework au choix).

Le menu en haut de page (modèle ASP.NET MVC de base) offrira un lien pour accéder à chaque exercice.

L'objectif est de pratiquer les concepts d'ASP.NET MVC : Modèles, Vues et Contrôleurs au travers d'exercices pratiques et de restituer un maximum de connaissances acquises durant les 3 jours de formations.

Le support de cours n'est donc pas autorisé. Internet non plus !

En revanche, tous les outils de génération et modèles de projet de Visual Studio peuvent être utilisés comme source d'inspiration pour tous les exercices.

Le livrable attendu pour chaque exercice est un code source ASP.NET MVC conforme aux exigences décrites dans l'énoncé, **documenté** et idéalement **testé**.

Les exercices n'ont pas de liens directs entre eux. L'ordre de réalisation n'a donc pas d'importance, toutefois il est préférable de commencer avec l'exercice 3 plus long et plus difficile…

Score :

Les deux premiers exercices sont sur 5 points chacun.

Le troisième et dernier exercice est sur 10 points.

Durée :

½ journée

Exercice 1 – Affichage d'une phrase avec une taille de police incrémentale (5 points) :

Objectifs :

- Ecrire un formulaire avec les HTML Helpers et/ou Tag Helpers
- Valider le modèle associé dans un contrôleur

Enoncé :

Ecrire une vue capable d'afficher un nombre prédéfini de fois une phrase – elle aussi prédéfinie – de manière que chaque fois que la phrase soit écrite, la taille de la police augmente (en démarrant à **12px**).

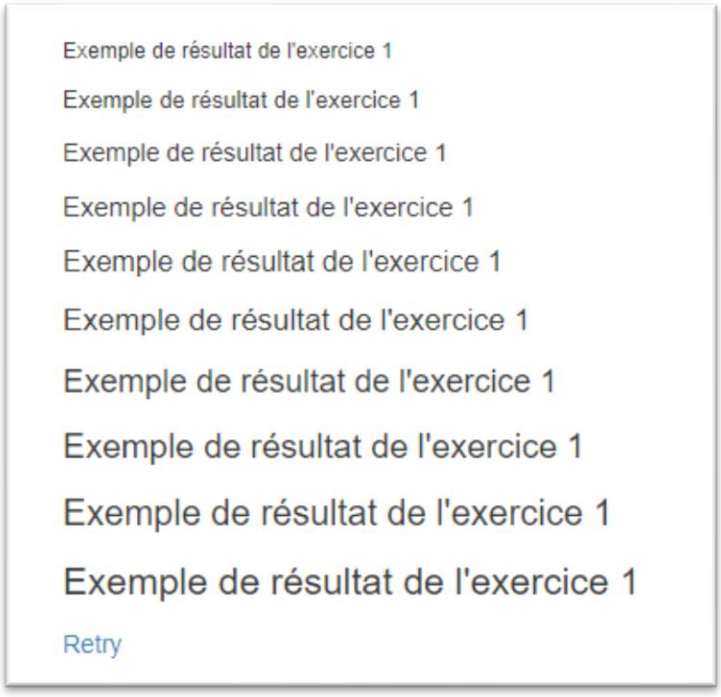
Afin de configurer cet affichage, il faudra décrire un formulaire permettant de saisir :

- Une phrase comprise entre **10** et **50** caractères (obligatoire)
- Un nombre d'écriture compris entre **5** et **20** (non obligatoire avec une valeur par défaut à 10)

Le formulaire devra conserver les valeurs en cas d'erreur et afficher des messages explicites sur les champs concernés.

Un lien permettra depuis la vue « Résultat » de retenter l'expérience avec une nouvelle phrase.

Exemple de résultat :



Contraintes :

Le formulaire devra être conçu en pensant à l'expérience utilisateur, c'est-à-dire facile d'utilisation et clair en termes de valeurs attendues.

Bonus :

Le contrôleur est couvert par des tests unitaires (cas de succès, cas d'erreur, etc.)

Exercice 2 – Convertisseur de devises (5 points) :

Objectifs :

- Ecrire un formulaire avec les HTML Helpers et/ou Tag Helpers
- Manipuler la validation du modèle associé dans un contrôleur
- Faire appel à une API externe

Enoncé :

Ecrire une vue capable d'afficher le résultat d'une conversion de devise.

Afin de configurer cet affichage, il faudra décrire un formulaire permettant de saisir :

- Le montant en devise à convertir (obligatoire)
- La devise du montant saisi parmi une liste de devises disponibles (obligatoire)
- La devise vers laquelle convertir parmi une liste de devises disponibles (obligatoire)

Le formulaire devra conserver les valeurs en cas d'erreur et afficher des messages explicites sur les champs concernés.

Pour cet exercice vous allez utiliser une API permettant de récupérer les taux de change (cf. Indices), si une erreur survient lors de la récupération ou que le taux de change est introuvable, une erreur doit apparaitre sur le formulaire pour mettre en évidence le dysfonctionnement.

Un lien permettra depuis la vue « Résultat » de retenter l'expérience avec une nouvelle configuration pour la conversion.

Exemple de résultat :

Contraintes :

Le formulaire devra être conçu en pensant à l'expérience utilisateur, c'est-à-dire facile d'utilisation et clair en termes de valeurs attendues.

Bonus :

Le contrôleur est couvert par des tests unitaires (cas de succès, cas d'erreur, etc.)

Indices :

Cf. Annexe 1 – Indices exercice 2

Exercice 3 – Enregistrement d'utilisateurs en base de données (10 points) :

Objectifs :

- Ecrire un formulaire avec les HTML Helpers et/ou Tag Helpers
- Valider le modèle associé dans un contrôleur
- Manipuler le modèle dans un contrôleur
- Enregistrer et récupérer des données depuis une base de données

Enoncé :

A l'aide d'un formulaire, permettre à un utilisateur de s'enregistrer sur le site.

Ce formulaire permettra de saisir ces champs (tous obligatoires) :

- Une adresse email
- Un mot de passe et sa confirmation (identiques) respectant les contraintes suivantes :
    - Au moins 8 caractères
    - Au moins une lettre MAJUSCULE
    - Au moins une lettre minuscule
    - Au moins un chiffre
    - Au moins un caractère spécial (@ # & % ! = ? + - * /)
    - Ne doit pas contenir le contenu de l'adresse mail (avant l'@)
    - **Exemple** : Thomas.Guillier123@ ne conviendra pas pour l'adresse mail thomas.guillier@sogeti.com
- Une carte de crédit :
    - Numéro de la carte (cf. règles ci-après)
    - Date d'expiration au format (MM/AA)
    - Le cryptogramme ne sera pas demandé avant un potentiel paiement

Le formulaire devra conserver les valeurs en cas d'erreur et afficher des messages explicites sur les champs concernés.

Lors de la soumission du formulaire et avant l'enregistrement en base de données, le contrôleur doit déterminer le type de carte et l'inscrire au modèle selon les règles suivantes :

Type de carte	Préfixe	Nombre de chiffre	Exemple
MasterCard	De 51 à 55	16	5109210173049302
Visa	4	13 ou 16	4012829173921881
American Express	34 ou 37	15	371449635398431

Après l'enregistrement dans la base de données, l'utilisateur devra être redirigé vers une vue mentionnant l'ID généré en base ainsi que l'ensemble des informations de la fiche utilisateur (email, type et numéro de la carte, date d'expiration).

La librairie pour l'accès aux données est libre (ADO.NET, Entity Framework, Dapper, etc.), seul le résultat compte.

Une base de données locale devra être utilisée pour l'exercice : (localdb)\MSSQLLOCALDB voire stockée en mémoire pendant la durée de vie de l'application, le choix est libre.

Contraintes :

Le formulaire devra être conçu en pensant à l'expérience utilisateur, c'est-à-dire facile d'utilisation et clair en termes de valeurs attendues.

Bonus :

Ajout d'un champ « Type de carte » grisé et sélectionné automatiquement en fonction de la saisie du numéro (utilisation de JavaScript dans la vue).

Le contrôleur est couvert par des tests unitaires (cas de succès, cas d'erreur, etc.)

Toute autre folie de votre imagination (à documenter !)

Indices :

Cf. Annexe 2 – Indices exercice 3

Annexe 1 – Indices exercice 2 :

Code source minimum pour la récupération du taux de change depuis l'API :

```
var httpClient = new HttpClient();
var response =
httpClient.GetAsync($"https://api.exchangeratesapi.io/latest?base={configuration.From.ToString()}
").Result;

var contentString = response.Content.ReadAsStringAsync().Result;
var response = JsonConvert.DeserializeObject<CurrencyRatesResponse>(contentString);

if (response.Rates.TryGetValue(configuration.To.ToString(), out decimal rate))
{
    // La variable "rate" contient le taux de change recherché.
}
```

La méthode **JsonConvert** provient du package NuGet **Newtonsoft.Json**.

Code de la classe « **CurrencyRatesResponse** » :

```
public class CurrencyRatesResponse
{
    public string Base { get; set; }
    public Dictionary<string, decimal> Rates { get; set; }
}
```

HTML Helper permettant de générer une « **SelectList** » depuis une énumération :

```
Html.GetEnumSelectList<EnumType>()
```

Annexe 2 – Indices exercice 3 :

Classes potentiellement utiles associées au namespace « System.ComponentModel.DataAnnotations » :

CompareAttribute	Fournit un attribut qui compare deux propriétés.
CreditCardAttribute	Spécifie qu'une valeur de champ de données est un numéro de carte bancaire.
CustomValidationAttribute	Spécifie une méthode de validation personnalisée qui est utilisée pour valider une instance de classe ou de propriété.
DataTypeAttribute	Spécifie le nom d'un type supplémentaire à associer à un champ de données.
DisplayAttribute	Fournit un attribut à usage général qui vous permet de spécifier les chaînes localisables pour les types et membres de classes d'entité partielles.
DisplayFormatAttribute	Spécifie la manière dont les champs de données sont affichés et mis en forme par Dynamic Data ASP.NET.
EditableAttribute	Indique si un champ de données est modifiable.
EmailAddressAttribute	Valide une adresse de messagerie.
EnumDataTypeAttribute	Permet à une énumération .NET Framework d'être mappée à une colonne de données.
KeyAttribute	Indique une ou plusieurs propriétés qui identifient une entité de manière unique.
MaxLengthAttribute	Spécifie la longueur maximale du tableau ou des données de type chaîne autorisée dans une propriété.

MinLengthAttribute	Spécifie la longueur minimale du tableau ou des données de type chaîne autorisée dans une propriété.
PhoneAttribute	Spécifie qu'une valeur de champ de données est un numéro de téléphone bien formé.
RangeAttribute	Spécifie les contraintes de plage numérique pour la valeur d'un champ de données.
RegularExpressionAttribute	Spécifie qu'une valeur de champ de données dans Dynamic Data ASP.NET doit correspondre à l'expression régulière spécifiée.
RequiredAttribute	Spécifie qu'une valeur de champ de données est obligatoire.
StringLengthAttribute	Spécifie les longueurs minimale et maximale de caractères autorisées dans un champ de données.
TimestampAttribute	Spécifie le type de données de la colonne en tant que version de ligne.
UrlAttribute	Fournit la validation de l’URL.