**ASP.NET MVC CORE 3.1**

**Documentation du projet  SportsStore**

**2020-2021**

Nom et prénom de l’étudiant : Barman Rayan

# Configuration

La classe « **Startup** » est responsable de la configuration de l’application. Elle a deux méthodes :

* ConfigureServices () : cette méthode est utilisée pour configurer les « services » de l’application. Un service est un composant réutilisable qui fournit des fonctionnalités d’application. Les « services » sont utilisées dans l’application grâce à l’injection de dépendance.
* Configure () : Cette méthode permet de configurer les middlewares. Les composants middleware permettent de traiter les requêtes http et même de produire la réponse à la requête. L’application .NET reçois des requêtes http qui les passes ensuite à un « Request pipeline » qui contient les composant middleware.

**Razor** est responsable du traitement des fichiers «.cshtml », il permet de générer du code HTML qui est compris par le navigateur.

**L’Entity Framework Core** fournit un accès à la base de données grâce à une classe « context » qui représentera les données dans la base de données.

On configure aussi dans la classe Startup la connexion à la base de données grâce à l’objet de type IConfiguration, nous avons accès au fichier json dans laquelle nous pouvons récupérer le nom de la base de données.

Le type **IConfiguration** permet d’avoir accès au système de configuration d’ASP .NET Core.

Création de la base de données sur base des modèles grâce à la fonctionnalité migration. Un fichier sera créé avec le code Sql qui permet de créer la base de données.

Quand ASP.NET Core a besoin de créer une nouvelle instance du HomeController pour traiter les requêtes http, il inspectera le constructeur et verra qu’il a besoin d’un objet de type IStoreRepository. Pour déterminer l’implémentation de l’objet, ASP.NET Core consulte la configuration dans la classe Startup, qui lui donne l’implémentation à utiliser. Ainsi chaque requête créera une nouvelle instance de EFStoreRepository. On utilise ici le mécanisme de l’injection de constructeur car on utilise l’interface et pas l’implémentation directement. Comme une interface ne doit jamais, on aura aucun changement si on change d’implémentation.

Une « **view model class** » est une classe spécifique pour passer des données entre un contrôler et une vue. Dans notre projet la classe qui fait ce travail est : **ProductsListViewModel**. Elle contient une collection de produits, …

Pour permettre au client de naviguer à travers les pages d’une liste de produits sans devoir mettre dans l’Url la page qu’on souhaite, nous mettons en places un **Tag Helper** qui permettra de naviguer de pages en pages grâce à un système de « lien » avec la balise <a>.

Le back-end à différents endpoints qui permet d’accéder à une ressource que l’on souhaite. Pour chaque endpoint il faut définir son URL ainsi que le contrôler qui est rattacher à ce endpoint et la méthode du contrôler. C’est cette méthode qui s’occupera de traiter la requête http venant sur ce endpoint. Le client envoie une requette http en spécifiant le endpoint grâce à ….

Un **view components** est un composant qui peut être réutiliser plusieurs fois. Comme nous voulons que le menu de navigation apparaisse sur plusieurs pages nous le mettons dans un composant réutilisable.

**Razor Pages** est un Framework alternatif au Framework MVC. Tout d’abord il faut configurer Razor Pages. Pour cela il faut aller dans la classe startup et rajouter Razor Pages aux services et le rajouter comme endpoint pour qu’on puisse accéder à la page comme celle-ci ne sera lier à aucun contrôleur. C’est elle-même qui traite les requêtes. Razor pages à ses propres fichiers de configuration dont son propre layout. Un Razor page se distingue par une mention @page tout au-dessus. La page razor à une « page model class » qui lui est associé. Cette classe permet de traiter les requêtes http qui arrive sur l’url de la page razor.

Activer une session quand on sur l’application pour ne pas perde ses données : Pour activer une session dans l’application il faut se rendre dans le fichier Startup et y ajouter dans la méthode ConfigureServices AddDisrtibutedMemoryCache() qui permet de mettre en place le stockage des données en mémoire. La méthode AddSession() permet d’accéder aux données de la session. UseSession() permet d’associer les requêtes du client à une seul session.