# Todo List – partie 1

**Objectifs**

* Créer une première appli web et la lancer depuis Visual Studio et en ligne de commande
* Créer et configurer correctement un contexte de données
* Générer un contrôleur et des vues de façon automatique
* Utiliser les différents types de requêtes http et les attributs sur les actions

L’objectif de l’application est de gérer une petite liste de tâches. Les données seront stockées dans une base de données générée dynamiquement par EF Core dans l’approche Code First.

**Etape 1 : Création du projet et lancement de l’appli**

* Créer un nouveau projet nommé TodoList, de type « Application web ASP.Net Core » avec l’architecture MVC.
* Lancer manuellement la partie serveur de l’application au moyen d’une ligne de commande, puis la partie cliente en recopiant l’url de la page d’accueil dans un navigateur.
* Lorsqu’on navigue parmi les trois pages de l’application, observer ce qui s’affiche dans la fenêtre console (nom des actions déclenchées), ainsi que dans la barre d’adresse du navigateur.
* Arrêter l’application serveur dans la fenêtre console en tapant CTRL+C, puis fermer la page dans le navigateur
* Lancer maintenant l’application en debug depuis Visual Studio
* Mettre un point d’arrêt dans la méthode Index de la classe HomeController, pour voir à quel moment on passe dans cette méthode.
* Fermer la page dans le navigateur. Que constate-t-on dans Visual Studio ? Arrêter le debug depuis Visual Studio

**Etape 2 : Création du modèle**

Nous allons créer un modèle Entity Framework très simple, dans l’approche CodeFirst sans base de données existante. Pour cela :

* Ajouter les paquets pour Entity Framework Core à l’aide de la ligne de commande du gestionnaire de package NuGet.
* Dans le dossier Models, créer une classe POCO nommée Tache avec les propriétés suivantes :  
  Id (entier), Description, DateCreation, DateEcheance, Terminee (booléen)

**Etape 3 : Création et configuration du contexte de données**

* Créer une classe TodoListContext dérivée de DbContext, et contenant un DbSet de tâches
* Dans le fichier appsettings.json, ajouter une chaîne de connexion vers une base TodoList, qui sera créée dynamiquement
* Dans la classe Startup, enregistrer le contexte dans la liste des services, et lui passer la chaine de connexion
* Ajouter un constructeur au contexte, afin qu’il puisse recevoir les options de configuration, contenant la chaîne de connexion

**Etape 4 : Création de la base de données**

* Générer la base de données à partir du modèle créé précédemment, en utilisant les commandes Add-Migration et Update-Database du gestionnaire de package NuGet.
* Dans SSMS, vérifier que la base est bien créée et qu’elle contient une table Tache
* Supprimer la table d’historique des migrations dans la base, et le dossier Migrations dans la solution, car nous ne gérerons pas de mise à jour de la base à partir du modèle

**Etape 5 : Création d’un contrôleur**

Dans le dossier « Controllers », ajouter un contrôleur de type « Contrôleur MVC avec vues utilisant EF », qui utilise la classe Tache comme modèle et le contexte de données qu’on vient de créer. Laisser la case « Générer des vues » cochée.

**Etape 6 : Ajout d’un menu et test de l’application**

* Ajouter une entrée dans le menu principal de l’application en éditant le fichier \_layout.cshtml
* Lancer l’application et vérifier que le menu et la vue Index de gestion des tâches s’affichent bien
* Tester l’application en créant, modifiant et supprimant des tâches

**Etape 7 : Analyse du code généré automatiquement**

* Observer la classe TachesController générée par VS. Pourquoi les actions Create, Edit et Delete sont-elles toutes en deux exemplaires ?
* Par quelles vues sont appelées les différentes actions de ce contrôleur ?

**Etape 8 : Modification de l’action Contact**

* Dans la classe HomeController, ajouter un paramètre « id » de type entier à l’action Contact, et modifier son code pour que la vue affiche la valeur du paramètre
* Lancer l’application depuis VS par CTRL + F5, et tester en passant une valeur d’id dans l’url de la page Contact. Quels sont les deux formats d’url possibles pour passer cet id ?
* Ajouter un second paramètre « nom » à l’action Contact, et faire en sorte que la vue affiche « Bonjour xxx, ton id est yyy »
* Sans recompiler l’application, tester cette modification en saisissant une url adéquate dans le navigateur. Quels sont les deux formats d’url possibles ?

**Etape 9 : attributs**

* Faire en sorte que l’action Contact n’accepte que les requêtes http POST. Tester puis revenir au comportement précédent
* Faire en sorte que l’action Contact ne puisse être appelée que par le nom « ContactUs » dans l’url de la requête, et que la page correspondante s’affiche toujours correctement.

**Etape 10 : Formatage des données**

Faire en sorte que la date de création et la date d’échéance soient affichées sous un format ordinaire (jj/mm/aaaa)

**Etape 11 : Validation des données**

Notez qu’en l’état actuel, il est possible de saisir une tâche avec une description vide. La saisie de la date d’échéance est par contre obligatoire.

On souhaite que :

* La description soit obligatoire, avec une longueur maximale de 250 caractères
* La date d’échéance soit facultative

Ajouter les contraintes nécessaires sur le modèle et recréer la base de données pour que la table reflète bien les nouvelles contraintes.

Supprimer la table d’historique des migrations dans la base, et le répertoire Migrations dans la solution.

**Etape 12 : Utilisation de Html Helpers et Tag Helpers**

Dans la vue Edit, remplacer le Tag Helper par un HTML Helper pour la case à cocher Terminée. A vous de trouver le HTML Helper adéquat (cf. doc Microsoft dont le lien est donné dans le cours si besoin).

Faire en sorte que la description de la tâche s’affiche sur plusieurs lignes dans les vues de création, d’édition et de détail de la tâche.

# Todo List – partie 2

**Objectifs :** se familiariser avec les notions suivantes

* Edition des vues
* Passage de données à la vue
* Liaison de modèle
* Formats d’affichage
* HTML Helpers et Tag Helpers

On se propose d’implémenter une page utilitaire, dont le visuel ressemble à ceci :



Il s’agit d’une calculatrice permettant de calculer une date à partir d’une date initiale et d’un nombre de jours qu’on ajoute. Le résultat s’affiche lorsqu’on clique sur le bouton =.

**Etape 1 : Création du contrôleur de la vue et du modèle**

Créer un nouveau contrôleur MVC vide nommé UtilitairesController

Créer une vue vide nommée Index et associée à ce contrôleur.

Créer une vue **partielle** vide nommée CalculsDates dans le même dossier. Ajouter une référence à cette vue dans la vue Index.

Ajouter une entrée Utilitaire dans le menu principal de l’application pour appeler l’action par défaut du contrôleur.

Créer une entité de modèle nommée Calcul qui représentera les données nécessaires à l’utilitaire

**Etape 2 : Code de la vue**

Dans la vue Index, implémenter le visuel défini plus haut en utilisant le positionnement et les styles BootStrap, et un formulaire contenant des éléments input et label. S’inspirer du code d’une autre vue générée automatiquement par Visual Studio et de [cette page d’aide BootStrap](https://v4-alpha.getbootstrap.com/components/forms/#inline-forms) pour la disposition d’un formulaire en ligne.

Utiliser des attributs sur les propriétés de l’entité Calcul pour que les zones de saisies s’adaptent aux types de données.

**Etape 3 : Action du contrôleur**

Dans le contrôleur, ajouter une action nommée AjouterJours qui prend en paramètres un objet Calcul. Faire en sorte qu’elle génère sa réponse avec la vue Index, en lui passant l’objet Calcul dans lequel on aura renseigné le résultat de l’opération.

Dans l’action Index, instancier une entité Calcul permettant d’initialiser la date de la première zone de saisie à la date du jour. Générer la réponse de la même façon que dans AjouterJours

Tester : on constate que le résultat ne s’affiche pas.

Ajouter le code suivant au début de l’action AjouterJours :

ModelState.Remove("DateRes");

Tester de nouveau : cette fois le résultat s’affiche.

Explications : ModelState représente l'état de la liaison de modèle (liaison entre la source de données de la vue et les paramètres de l’action) après saisie des valeurs par l'utilisateur. Il doit être rafraîchi pour que le résultat de l'opération s'affiche. On peut forcer ce rafraichissement en vidant le ModelState complètement ou seulement la propriété correspondant à la donnée à rafraîchir.

**Etape 4 : Passage du résultat avec ViewBag**

Faire en sorte que le résultat de l’opération soit désormais transmis à la vue par le ViewBag et non plus par l’entité Calcul. On mettra le code créé précédemment en commentaire pour pouvoir y revenir par la suite.

Indication : dans la vue, utiliser l’attribut value au lieu de asp-for pour afficher la valeur.

Faire en sorte que le résultat s’affiche toujours selon un format correct, sans les heures.

**Etape 5 : Affichage du nombre de tâches terminées**

Dans l’action Index du contrôleur Taches, stocker le nombre de tâches terminées dans le dictionnaire ViewData, et dans la vue correspondante, afficher une libellé « X tâche(s) terminée(s) sur Y » sous le titre. Le nombre total de tâches sera récupéré directement sur le modèle. Les « s » à la fin des mots ne doivent s’afficher que lorsque c’est nécessaire

# Todo List – partie 3

**Objectifs :** se familiariser avec les notions suivantes

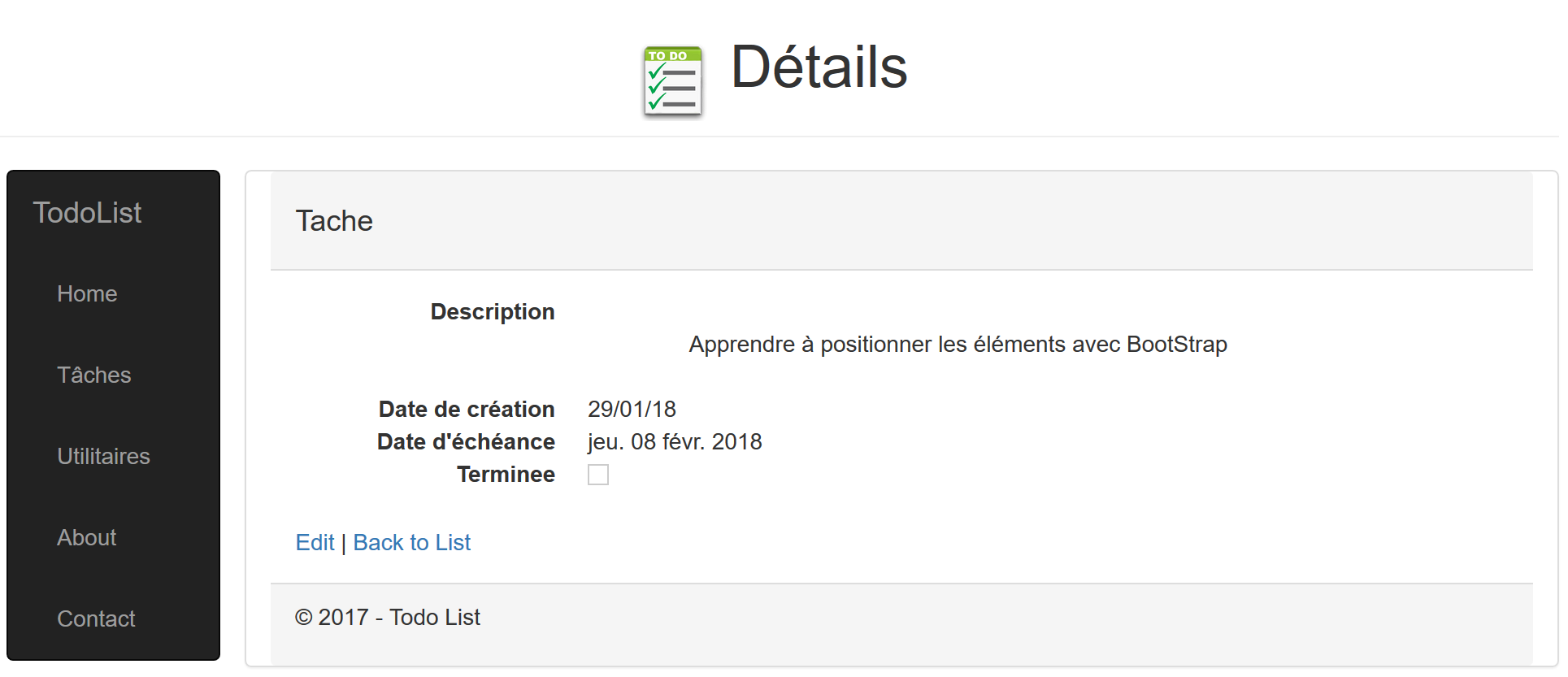
* Page de disposition, styles Bootstrap
* Formulaires
* Validation et formats d’affichage
* Vues modèles
* Tri et filtrage

**Etape 1 : Création d’une page de disposition spécifique**

Créer une page de disposition qui s’applique automatiquement à toutes les vues de gestion de tâches. Elle doit décrire une disposition ressemblant à ce qui suit :

|  |  |
| --- | --- |
| Titre | |
| Menu vertical | Sous-titre |
| Corps |
| Pied de page |

Exemple de rendu pour la page de détails :



L’image du titre est donnée ci-dessous :



**Indications**

* Reprendre les éléments <environnement> de la page de disposition par défaut, de façon à pouvoir utiliser la bibliothèque Bootstrap
* Le style du titre est obtenu avec la classe de style page-header de Bootstrap
* Toute la zone à droite du menu est contenue dans un panel possédant un titre, un corps et un pied. Utiliser pour cela les classes de style panel, panel-heading, panel-body et panel-footer de Bootstrap

**Etape 2 : Modification des vues**

* Modifier les vues de tâches pour qu’elles s’intègrent bien dans la disposition créée précédemment.
* Dans les vues de création et de modification, mettre le sous-titre « Tâche »
* Dans la vue de suppression, mettre le sous-titre « Confirmez-vous la suppression de cette tâche ? »
* Dans la vue liste, mettre le lien « créer une nouvelle tâche » dans le coin supérieur droit du tableau
* Mettre tous les libellés en français

**Etape 3 : Choix de l’opération dans la calculatrice de dates**

Dans le formulaire de calcul de dates, remplacer le signe « + » par une liste déroulante permettant de choisir l’opération à effectuer (addition ou soustraction). Pour cela, utiliser un élément <select> contenant des éléments <option>, comme dans l’exemple suivant :

<**select** **asp-for**="CountryCode">

    <**option** **value**="CA">Canada</**option**>

    <**option** **value**="US">US</**option**>

    <**option** **value**="--">Other</**option**>

</**select**>

Modifier l’entité de modèle et l’action de contrôleur en conséquence.

**Etape 4 : Validation des données dans l’utilitaire de calcul de dates**

Faire en sorte qu’on ne puisse saisir qu’un nombre de jours compris entre 1 et 9999. Afficher un message d’erreur explicite si c’est le cas.

**Etape 5 : Validation des données sur les tâches**

Créer une règle de validation côté serveur pour vérifier que la date d’échéance est supérieure à la date de création de la tâche.

**Etape 6 : Filtrage des tâches**

Dans la vue de la liste des tâches, entre le titre et le tableau, ajouter les éléments de filtre suivants :

* Zone de texte pour filtrer les tâches dont la description contient le texte saisi
* Liste déroulante pour filtrer les tâches selon leur état (terminée ou non)
* Bouton Filtrer

Les valeurs de filtre saisies devront être restaurées après application du filtre.

**Etape 7 : Routage par attributs**

Supprimer le routage conventionnel défini dans la méthode Startup.Configure, et au moyen d’attributs de routage, faire en sorte que les actions des différents contrôleurs puissent continuer d’être appelées par les mêmes url. Utiliser le minimum d’attributs possible.

Faire en sorte qu’on puisse appeler l’utilitaire de calcul de dates par son url actuelle, et aussi par l’url « localhost:xxxx/calculatrice »

# Base Ciqual des aliments

**Objectifs** : Mettre en œuvre les notions suivantes

* Modèle de données Entity Framework Core avec base de données existante
* Syntaxe Razor pour éditer les vues
* Echange de donnée entre vue et contrôleur
* Vues maître-détail simple et hiérarchique
* Modèles de vues

On souhaite créer une application web permettant d’interagir avec la base Ciqual des aliments, fournie par l’ANSES. Le schéma de cette base est le suivant :

Une image contenant capture d’écran

Description générée avec un niveau de confiance très élevé

**Etape 1.1 : Création du projet**

Créer un nouveau projet ASP.Net MVC Core nommé Ciqual, en activant l’authentification d’utilisateur individuel

**Etape 1.2 : Connexion à la base**

* Remonter la sauvegarde de la base de données qui vous sera fournie
* Modifier le nom de la base dans la chaîne de connexion présente dans le fichier appsettings.json

**Etape 1.3 : Création du modèle et configuration du contexte**

* Créer le modèle de données à partir de la base dans le répertoire Models en exécutant la commande fournie dans le fichier suivant dans le gestionnaire de package NuGet :



* Exécuter la commande « update-database -context ApplicationDbContext » dans le gestionnaire de package NuGet, afin d’ajouter les tables nécessaires à la gestion de l’authentification (ces tables sont définies dans la classe CreateIdentitySchema générée automatiquement par VS lors de la création du projet)

On ne souhaite garder qu’un seul DbContext dans l’application. Pour cela :

* Recopier les propriétés DbSet et le code de la méthode OnModelCreating de la classe CiqualDbContext dans la classe ApplicationDbContext.  
  Attention à bien garder l’appel à la méthode de base dans OnModelCreating !
* Supprimer la classe CiqualDbContext.
* Renommer le fichier et la classe ApplicationDbContext en CiqualDbContext dans toute l’application à l’aide du renommage automatique de VS
* Générer le projet pour vérifier que tout est bon

NB/ Lors de la génération du projet, VS a déjà ajouté le contexte à la liste des services de l’application (dans son conteneur IoC), et passé les options de configuration en paramètre à son constructeur

**Etape 1.4 : Menu et page d’accueil**

Dans la page de disposition Shared\\_Layout.cshtml, ajouter les entrées de menu qui permettront d’accéder aux pages de l’application :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ciqual | Accueil | Familles | Constituants | Aliments |

NB/Les entrées de menu seront câblées sur des actions de contrôleur au fur et à mesure de leur création.

Modifier la page d’accueil pour qu’elle affiche le contenu suivant :

139 familles

61 constituants

2642 aliments

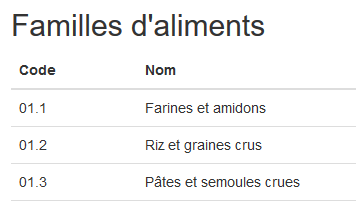
NB/ Ce visuel sera amélioré à la fin de cet exercice.

**Etape 1.5 : Création du contrôleur Familles**

* Créer un contrôleur nommé FamillesController permettant d’afficher la liste des familles d’aliments. Ne pas générer les vues automatiquement, car nous n’utiliserons qu’une seule vue.
* Après génération de la classe par Visual Studio, ne garder que l’action Index, et supprimer les autres

**Etape 1.6 : Création de la vue des Familles**

* Dans un sous-répertoire « Views \ Familles », ajouter une vue nommée Index selon le modèle « List ».
* Après génération, faire en sorte qu’elle affiche ce qui suit :



Les libellés des en-têtes de colonnes seront définis à partir d’attributs sur les propriétés de l’entité Famille.

* Supprimer les liens Create new, Edit, Details et Delete générés automatiquement.
* Faire en sorte que cette vue soit appelée lorsqu’on clique sur le menu « Familles » de la page d’accueil.

**Etape 1.7 : Création de la page des constituants**

* Créer de la même façon une page affichant la liste des constituants, sans leurs id et triés par ordre alphabétique de leur nom.
* Supprimer les méthodes inutiles dans le contrôleur, ainsi que les éventuelles vues inutiles.

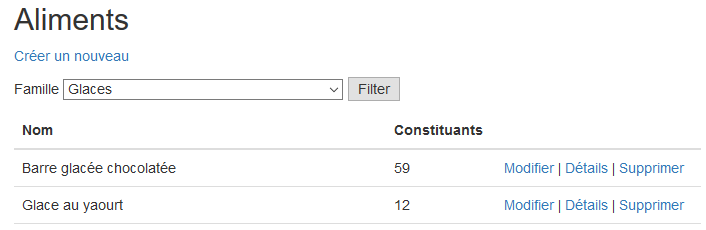
**Etape 1.8 : Tri des Familles**

Implémenter le tri des familles lorsqu’on clique sur les entêtes de colonnes. Le tri doit être alternativement ascendant puis descendant.

Pour cela, mettre des liens sur les libellés des en-têtes de colonnes en utilisant des Tag Helpers.

**Les étapes qui suivent concernent l’affichage des aliments par famille**

Comme il y a plus de 2600 aliments dans la base, on ne veut pas afficher la liste complète d’un coup, mais plutôt afficher les aliments par famille, comme le montre l’affichage suivant :



Pour créer cette vue, suivez les étapes suivantes :

**Etape 2.1 : Création d’une entité visuelle**

* Créer une entité visuelle AlimentsVM agrégeant les informations nécessaires à cette vue.
* Ajouter des attributs sur les propriétés d’entités pour définir leurs libellés d’affichage.

**Etape 2.2 : Création du contrôleur Aliments**

* Générer un contrôleur Aliments pour l’entité Aliment, en générant les vues en même temps.
* Modifier la méthode Index du contrôleur, pour :
  + Ajouter un paramètre correspondant au code de la famille sélectionnée
  + Récupérer les aliments appartenant à cette famille
  + Créer et initialiser une entité AlimentsVM, puis la passer à la méthode View().

**Etape 2.3 : Création de la vue regroupée par famille**

Modifier la vue Index de la façon suivante :

* Affecter l’entité visuelle comme modèle de la vue
* Ajouter un formulaire contenant la liste déroulante et le bouton Filtrer. Par défaut, le premier élément doit être sélectionné dans la liste déroulante
* Modifier le code Razor pour que le tableau affiche les éléments de la propriété Aliments de la vue modèle
* La colonne du nombre de constituants sera gérée à l’étape suivante. Ne pas la créer tout de suite

NB/ Il se peut que les caractères accentués soient mal retranscrits dans la page. Ceci vient du fait que Visual Studio enregistre le fichier de la vue dans le jeu de caractères ANSI. Pour remédier à ce problème, ouvrir le fichier cshtml dans Notepad++, modifier son encodage en UTF-8, corriger les caractères mal retranscrits, et enregistrer le fichier.

**Etape 2.4 : Affichage du nombre de constituants de chaque aliment**

* Ajouter une colonne dans le tableau des aliments pour afficher le nombre de constituants. Ce nombre sera obtenu en comptant le nombre d’items de la collection « Composition » de la propriété Aliment.
* Tester. Par défaut, la valeur affichée est 0, pourquoi ? Faire en sorte d’obtenir la bonne valeur.
* La vue Index ne fait qu’afficher les données, sans les modifier. Pour optimiser les performances, désactiver le suivi des modifications sur le DbSet Aliment.

**Les étapes qui suivent concernent l’affichage du détail d’un aliment**

On souhaite obtenir le visuel suivant pour le détail d’un aliment :



La partie supérieure affiche le nom et la famille de l’aliment, tandis que la partie inférieure affiche sa composition. Pour réaliser cette vue, suivez les étapes ci-dessous :

**Etape 3.1 : Modification de l’action du contrôleur**

Modifier la requête de l’action Details() du contrôleur Aliments, afin qu’elle récupère la composition et les constituants de l’aliment. Pour cela, utiliser les méthodes Include et ThenInclude dans la requête Linq To Entities.

**Etape 3.2 : Modification de la vue**

Modifier la vue pour qu’elle ressemble à ce qu’on souhaite.

Indication : pour récupérer les libellés des en-têtes de colonnes, vous devrez modifier le type de collection de la propriété Composition de l’entité Aliment.

**Les étapes qui suivent concernent l’affichage de la liste d’aliments par lettres de l’alphabet**

On souhaite créer une vue supplémentaire où les aliments seront affichés selon la première lettre de leur nom. Le nom de la leur famille d’appartenance devra également être affiché :



Pour réaliser cette vue, suivez les étapes ci-dessous :

**Etape 4.1 : Création de la vue**

* Ajouter une nouvelle vue pour le contrôleur Aliments, nommée ListByFirstLetter, de type List
* Ajouter un bloc de code Razor permettant de générer les 26 liens hypertextes correspondant aux lettres de l’alphabet. On fera en sorte que la lettre cliquée soit passée au contrôleur par le segment de route id.
* Câbler cette vue sur le bouton correspondant de la page d’accueil

Indices pour créer la suite de lettres :

* Les lettres peuvent être générées à partir de leur code ASCII au moyen de la méthode Char.ConvertFromUtf32().
* La classe de style Bootstrap « list-inline » sur un élément <ul> permet d’obtenir une liste horizontale
* Le Tag Helper asp-route-xxx sur un lien, permet de spécifier la valeur d’un segment de route xxx.

**Etape 4.2 : Création de l’action dans le contrôleur**

* Dans le contrôleur, ajouter l’action ListByFirstLetter, capable de traiter le segment de route id
* Dans cette action, récupérer la liste des aliments dont la première lettre du nom correspond à la lettre cliquée, ainsi que leurs familles. Les aliments seront triés par nom.

**Etape 4.3 : Pagination**

Ajouter une pagination sur la liste des aliments pour les afficher par lots de 20. On naviguera entre les pages au moyen de boutons Précédent et Suivant placés en bas de page. Les boutons doivent être grisés lorsqu’on est respectivement sur la première et sur la dernière page.

Bonus : afficher les numéros de pages entre les boutons Précédent et Suivant pour un accès direct à chaque page :



**Etape 4.4 : Optimisation des performances de la liste des aliments**

La méthode utilisée pour afficher le nombre de constituants de chaque aliment nous oblige à charger la composition des aliments. Ce n’est pas optimal, car on charge en mémoire des informations inutiles, ce qui augmente la charge du serveur et allonge le temps d’affichage de la page.

Dans l’action Index, au lieu d’utiliser le DbSet des aliments, exécuter une requête SQL qui ramène uniquement les informations nécessaires.

**Etape 4.5 : Visuel de la page d’accueil**

Modifier la vue de la page d’accueil pour qu’elle ressemble à ceci :



L’image est donnée ci-dessous :



**Indications** :

* Les éléments de navigation sous le titre sont placés dans un élément <nav> contenant une liste de type : <ul class="nav nav-pills h3" role="tablist">
* Voir [cette page](https://www.w3schools.com/bootstrap/bootstrap_navbar.asp) pour faire un menu déroulant
* Noms des icônes dans les éléments de navigation : glyphicon-grain, glyphicon-tint, glyphicon-apple
* Les compteurs sont réalisés au moyen de : <span class="label label-primary">

**Etape 4.6 : Authentification**

Faire en sorte que l’authentification soit obligatoire pour ajouter, modifier ou supprimer un aliment.

## API Web

**Objectifs** : apprendre à :

* Créer une API Web
* Tester les méthodes avec PostMan
* Créer une application cliente

**Etape 1 : Création du projet**

Créer une application web ASP.Net Core, nommée CiqualAPI, et de type API web.

**Etape 2 : Création du modèle**

On veut exploiter la base Ciqual avec EF dans l’approche Code First avec base existante. Pour cela :

* Ajouter la chaîne de connexion dans le fichier appsettings.json
* Générer le modèle EF dans un dossier Models grâce à la commande scaffold, en prenant les 4 tables de la base
* Remplacer la méthode OnConfiguring par un constructeur qui prend la chaîne de connexion en paramètre, et la passe au constructeur de sa classe de base.
* Dans la méthode Startup.ConfigureServices, ajouter le DbContext à la liste des services, avec la chaîne de connexion récupérée dans Settings (idem exo Ciqual)
* Générer le projet pour vérifier que tout est bon

**Etape 3 : Création d’un contrôleur d’API**

* Ajouter un contrôleur d’API avec actions, utilisant EF pour la table Aliment
* Renommer la méthode d’API qui récupère l’ensemble des aliments en GetAliment**s**
* Exécuter l’application et tester la méthode d’API qui récupère un aliment à partir de son Id, en saisissant son url dans le navigateur
* Tester la méthode en lui passant un Id d’aliment qui n’existe pas. Que se passe-t-il ? Examiner la réponse avec les outils de développement du navigateur, dans la rubrique réseau. Quel es le code de statut de la requête http ?

**Etape 4 : Test de l’API avec PostMan**

Tester toutes les actions de l’API à l’aide de Postman et enregistrer les requêtes

**Etape 5 : Création d’une application cliente**

Créer une application console cliente de l’API Web, et permettant de tester toutes ses méthodes.

**Etape 6 : Simulateur de ration**

On souhaite créer un service qui facilite la création de rations alimentaires.

L’application cliente envoie au service une ration qui est constituée d’un ensemble d’aliments identifiés par leurs Id, avec des quantités en grammes. Elle récupère en retour les caractéristiques de la ration avec sa valeur énergétique et les quantités de lipides, glucides et protéines qu’elle contient.

Vous devez :

* Créer la méthode de service CalculRation
* La tester dans l’application cliente. Dans un premier temps, créer une ration en dur dans le code, et dans un second temps, la faire saisir à l’utilisateur