# 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

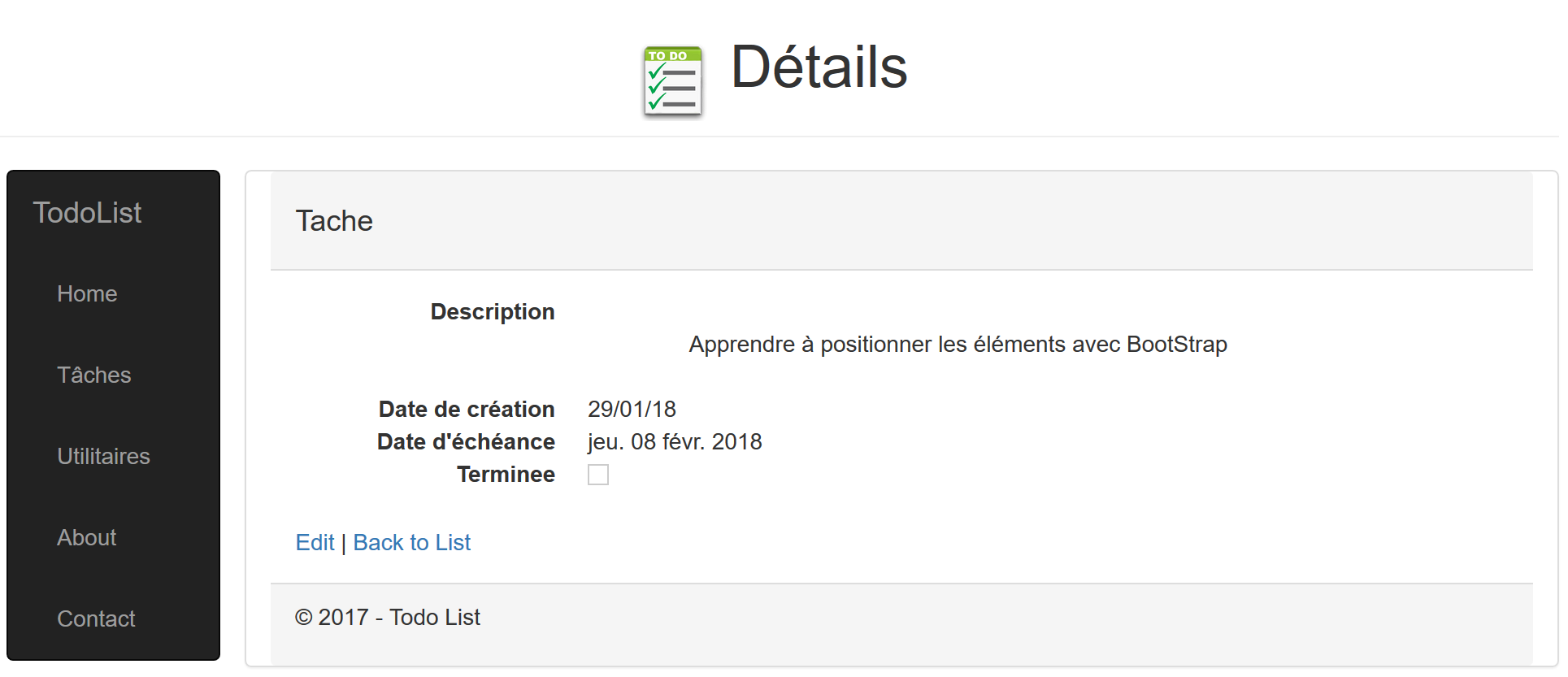
* Page de disposition, styles Bootstrap
* Formulaires
* Validation et formats d’affichage
* Vues modèles
* Tri et filtrage

**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.