**Laboratoire #11**

Gestion d’images avec Web API

**Projets de départ**

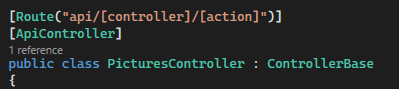
* Un serveur **.NET Web API** qui se nomme **ServeurImages** et qui va servir de base pour gérer une petite libraire d’images.
* Un client **Angular** qui se nomme **client-images** et qui contient déjà 2 components :
  + Le component **images** est utilisé pour afficher la liste des images en petit format et permettre d’ajouter et d’effacer des images de notre librairie.
  + Le component **singleimage** est utilisé pour afficher une image en haute résolution.

**Étape 1 - Ajouter les fonctionnalités de gestion d’image à votre serveur**

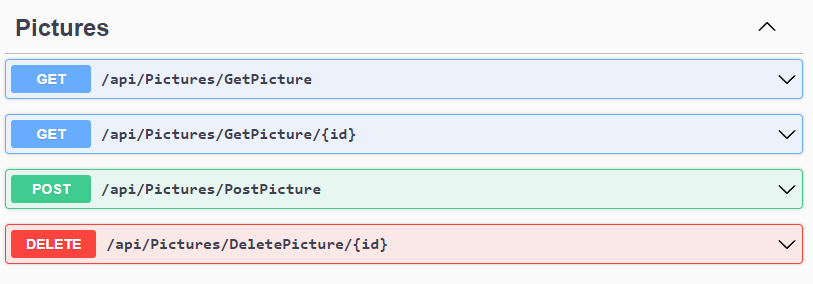
* 6  Complétez la classe **Picture**. C’est important qu’elle contienne les mêmes champs que la classe **Picture** de votre **client**. La seule différence devrait être l’utilisation de **majuscule** pour le premier charactère en C#. (PascalCase vs camelCase)



* 7  Créez un contrôleur **PicturesController** avec des actions auto-générées pour ce modèle.
  + Modifiez la route de votre contrôleur pour qu’elle contienne l’action.



* + On n’utilisera pas les méthodes **PutPicture**, alors vous pouvez simplement l’enlever.
  + Exécutez ensuite votre application et vérifiez que **Swagger** affiche les informations des requêtes de ce nouveau contrôleur.



* C’est maintenant le temps de faire votre migration et de mettre à jour votre DB avec **dotnet ef**
* **8-12**  Il faut maintenant modifier **PostPicture** pour gérer le fichier envoyé par le client comme il est mentionné dans les notes de cours.
  + Important! Dans ce cas, le client va nous envoyer un fichier à travers un formData et il n’est pas nécessaire pour le client d’envoyer une instance de Picture, ni au serveur d’en recevoir une. Vous pouvez modifier la fonction **PostPicture** pour qu’elle ne reçoive aucun paramètre.
    - Dans l’action, vous aurez à créer la « Picture » vous-mêmes avec **Picture picture = new Picture( ... );**
    - Dans les notes de cours, le *Birb* existe déjà, mais dans le labo, la picture doit être créée dans la même action que celle qui reçoit l’image.

Une image contenant texte

Description générée automatiquement

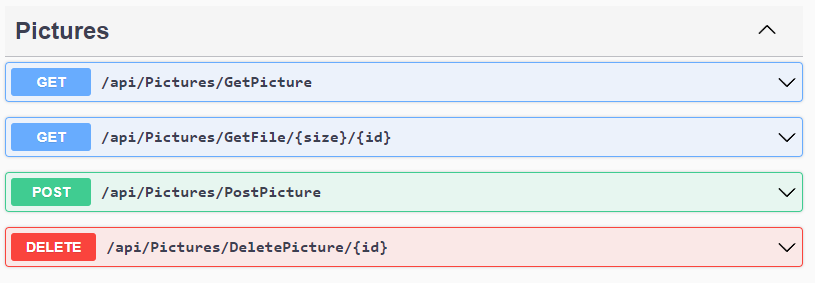
* + Utilisez **Request.ReadFormAsync** pour obtenir un **IFormFile**.
  + N’oubliez pas de désactiver la limite de taille pour la requête!
  + Pas besoin d’installer le package **ImageSharp**, il est déjà dans le projet.
* **12**  Continuez de compléter la méthode **PostPicture** pour créer une nouvelle instance de **Picture** et sauvegarder l’image sur disque. En fait, il faudra sauvegarder 2 images sur le disque, une petite et une grande en utilisant **Directory.GetCurrentDirectory()**.
  + Pour la grande image, simplement la sauvegarder pleine taille sous **/images/lg/**.
  + Pour la petite image, sauvegarder la avec une largeur maximale de 320 pixels sous **/images/sm/**.
  + Une fois que votre nouvelle instance est ajoutée à son **DbSet** et que les changements ont été sauvegardés, renvoyez-la au client avec **Ok()**.
  + IMPORTANT : On ne retourne pas le fichier d’image au client pour le moment. Nous allons bientôt modifier **GetPicture** pour que ça soit possible.

**Étape 2 - Ajouter une requête pour envoyer des images à votre serveur**

* Il est maintenant temps de faire un peu de travail sur le client. Comme toujours, on commence par rouler ***npm install*** sur notre projet Angular qui se nomme **client-images**.
* **5**  Pour ce laboratoire, nous utilisons la technique « Requête avec HttpClient et ViewChild » qui est expliquée dans les notes de cours.
  + Le client contient déjà un component **images** qui utilise le bon form, mais son **<input>** a besoin d’être configuré.
  + La fonction **uploadPicture()** doit faire une requête pour envoyer l’image sélectionnée au serveur.
  + Vous pouvez utiliser le nom de clé que vous voulez pour mettre le fichier dans le formData, mais c’est important d’utiliser le même nom du côté serveur au moment de lire les données.
  + Il ne faut pas oublier d’ajouter le **Picture** nouvellement créé à votre Array de **pictures** une fois que la requête est complétée.
* C’est maintenant le temps de tester! Vous devriez pouvoir utiliser le client pour ajouter une image au serveur. L’image ne s’affiche pas encore, mais les 2 fichiers devraient être créés au bon endroit et le serveur devrait retourner l’instance qu’il vient de créer.
* Vous pouvez également utiliser Swagger pour tester la méthode /api/Pictures/GetPicture. Assurez-vous qu’elle retourne bien un Array qui contient l’information sur les images que vous avez ajoutées.

**Étape 3 - Modifiez GetPicture****/{id} pour obtenir un fichier d’image**

* L’idée ici sera de retourner le fichier de l’image et non une instance de Picture.
  + Donc la première étape c’est de renommer la fonction pour **GetPicture/{id}** pour **GetFile/{id}**.
  + Ensuite il faut ajouter un nouveau paramètre pour la taille de l’image, donc on veut obtenir
    - **GetFile/{size}/{id}**
  + Vous devriez donc maintenant avoir l’API suivant :



* **13**  Il faut maintenant implémenter **GetFile**.
  + Utilisez **System.IO.FileReadAllBytes** pour obtenir le contenu du fichier.
  + Il faut ensuite utiliser la fonction **File** et retourner le fichier.
* **14**  Il ne reste plus qu’à implémenter la fonction **DeletePicture**. Assurez-vous d’effacer les fichiers d’images en plus d’effacer l’entrée de son **DbSet**.

**Étape 4 - Ajouter les fonctionnalités d’affichage et de suppression d’images au client**

* Tout d’abord, il faut obtenir l’information à propos des images. Ajoutez un appel dans **ImagesComponent.getPictures()** pour obtenir la liste des images.
* **15**  Pour l’affichage de vos images, ce n’est pas sorcier. Maintenant que vous avez une méthode d’api **GetFile/{size}/{id}**, il faut simplement spécifier la src de votre image pour qu’elle utilise celle-ci.
  + Il faut ajouter un élément <img> dans les templates des components **images** et **singleimage.**
  + Dans le template du component **images**, ajoutez également un **[routerLink]** au **<img>** qui va afficher l’image pleine taille en utilisant la route **/image/:id**.
  + Profitons de l’opportunité pour rafraîchir notre mémoire et récupérer l’id de la route dans **SingleImageComponent**.
* Il ne reste plus qu’à implémenter la fonction **ImagesComponent.deletePicture()** en ajoutant un appel http.delete.