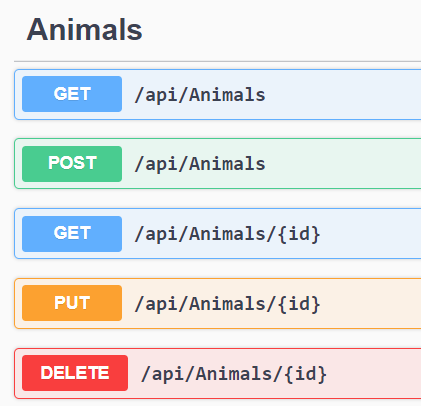
**Laboratoire #8**

Web API

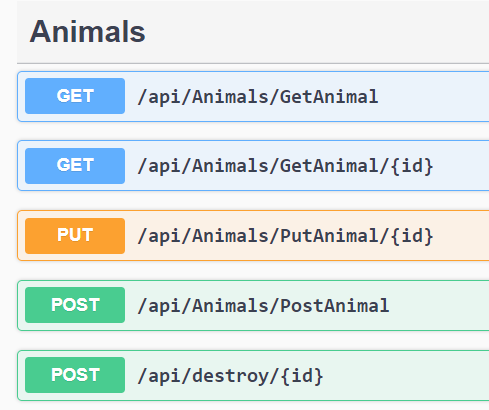
**Étape 1 - Mon premier projet Web Api 👶**

* ⭐ **8-10**  Créez un nouveau projet asp net core web avec le gabarit Web API. (Avec Core 6.0)
* 🗑 Supprimez le contrôleur et le modèle pour WeatherForecast : nous n’avons pas besoin de ces vulgaires classes prémâchées.
* 📂 **11-12**  Créez un nouveau Model (dans un dossier « Models » ! Nous ne sommes pas des animaux 🐒) nommé Animal.
  + Voici les propriétés d’un Animal :
    - Id (int) Ex. 2
    - Type (string) Ex. "Chien"
    - Name (string) Ex. "Maurice"
* **13**  Installez les trois packages nécessaires. (Entity Framework et CodeGeneration)
* ⚙ 1**5-18**  Créez un contrôleur avec des actions auto-générées pour ce modèle. Cela devrait installer les dépendances Entity Framework lorsqu’il y aura (peut-être) un message d’erreur.
  + Exécutez ensuite votre application et vérifiez que Swagger affiche les informations des requêtes de ce nouveau contrôleur.



**Étape 2 - Sur la bonne voie 🚗🗺**

* **21-29**  Changez la règle globale de routage de votre contrôleur pour que le nom des actions soit inclus dans les URL de requête.
* **21-29**  Ajoutez une règle spécifique à l’action DELETE pour que l’url de requête soit localhost:port/api/destroy
  + De plus, changez le type de requête pour POST. (Le contenu de l’action doit rester identique, cela servira encore à supprimer un Animal de la base de données. On s’amuse juste à changer le type de méthode / requête pour se rendre compte qu’on peut en faire ce qu’on veut)



**Étape 3 - Écrire du SQL c’est plate 😴**

* **19-20**  Créez la base de données pour votre Web API. Cela devrait être réglé avec quelques commandes simples. (Merci Entity Framework)

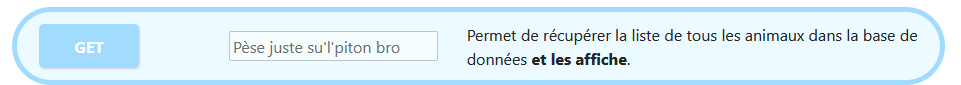
**Étape 4 - Abolir les frontières** ✂

* **35-36**  Assurez-vous que la Web API acceptera les requêtes de l’application Angular même si les deux applications n’utilisent pas le même port.
* Vous pouvez exécuter l’application Web API. Il n’y aura plus de modifications à faire avant un moment.

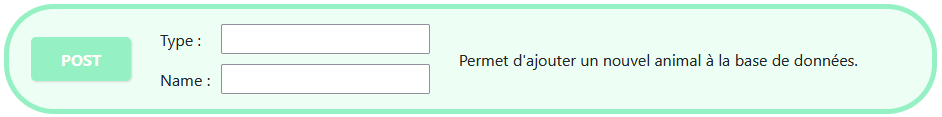
**Étape 5 - Angular ? Jamais entendu parlé**

* 📦 Réinstallez les dépendances du projet Angular fourni et exécutez-le.
* 📩 Nous allons implémenter les requêtes GET, POST, GET, PUT et DELETE, mais avant, nous avons quelques trucs à préparer :
  + 📦 **30, 32**  Importez le **module** nécessaire pour faire des requêtes HTTP. Injectez la **dépendance** nécessaire dans le composant app.
  + 🐒 **31**  Créez un **Model** qui représente un **animal**. (Calquez sur le Model d’animal dans le projet **Web API**)
  + Ajoutez deux variables de classe au composant App :
    - **animals** : un **tableau** « d’Animal » initialisé avec un tableau vide. [ ]
    - **animal**: un « Animal » possiblement **undefined**.
    - Dans le template HTML, servez-vous de la variable **animals** et d’un **\*ngFor** pour afficher la liste (pour le moment vide) dans la partie « **Liste d’animaux (Get 1)** ». (Affichez l’id, le type et le nom de chaque animal)
    - Dans le template HTML, servez-vous de la variable **animal** (pour le moment vide) pour afficher son id, son nom et son type dans la partie « **Animal (Get 2)** ». Vous aurez besoin de sécuriser l’affichage avec un **\*ngIf** car **animal** peut être undefined.

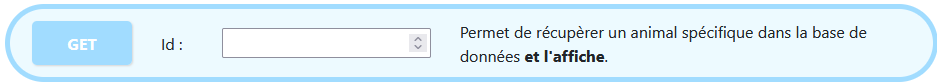
Pour les implémentations qui suivent, n’hésitez pas à jeter un coup d’œil à l’interface Swagger du Web API pour savoir quelles requêtes utiliser.



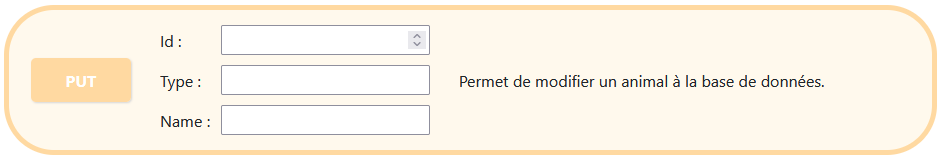
* Implémentation de **GET** (Fonction **getAnimals**)
  + **40-41**  Le but est de récupérer tous les animaux de la base de données, les stocker dans this.**animals** et de les afficher.



* Implémentation de POST (Fonction **postAnimal**)
  + **39**  En utilisant le Type et le Name spécifiés par l’utilisateur, créez un new Animal (pas dans la variable this.**animal** !) qui sera envoyé avec la requête. Son Id doit ABSOLUMENT être 0. (Entity Framework va lui assigner un Id convenable lui-même)



* Implémentation de GET (Fonction **getAnimal**)
  + **40-41**  En utilisant l’Id spécifié par l’utilisateur, remplissez la variable this.**animal** avec l’animal récupéré par la requête. (Cela devrait l’afficher dans la page) C’est normal que ça ne marche pas si l’id spécifié n’existe pas dans la base de données.



* Implémentation de PUT (Fonction **putAnimal**)
  + **42**  Créez un new Animal (Pas dans la variable this.**animal**) avec les informations spécifiées par l’utilisateur. (C’est normal que ça ne marche pas si l’id spécifié n’existait pas dans la base de données)
  + Envoyez cet animal dans le corps de la requête et n’oubliez pas de mettre l’id dans l’url comme pour le GET précédent.



* **42**  Implémentation de DELETE (Fonction **deleteAnimal**)
  + Bon... n’oubliez pas qu’on a donné le type POST à cette requête dans notre Web API. Ça veut dire qu’on est obligés de fournir un corps à la requête HTTP.
  + Cela dit, nous n’avons rien de pertinent à insérer dans le corps (Il suffit de mettre l’id de l’animal à supprimer dans l’URL...), donc mettez « *null* » comme corps.
  + Bien entendu, c’est normal que ça ne fonctionne pas si l’id spécifié n’existe pas dans la base de données. (Combien de fois devra-t-on vous le répéter ?)

On ne rend pas fonctionnelle la dernière section (« Buh-bye ») pour le moment.

Si tout marche, bravo 🥳 Vous avez une application serveur et une application client qui communiquent et sont « RESTful ». Il restera juste à ajouter une couche entre les contrôleurs et le DbContext pour améliorer l’encapsulation des opérations sur la BD... dans la partie 2.

**Laboratoire #8 : Partie 2**

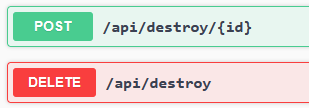
Services

N’oubliez pas qu’il faut refaire la commande dotnet ef database update quand on change le projet de machine !

**Étape 6 - Le projet fonctionnait très bien sans services 😠**

* **48-53**  Créez un service générique et « glissez-le » entre le DbContext et le contrôleur d’animaux.
  + Le service générique se fera injecter le DbContext et le contrôleur se fera injecter le service.
  + On veut simplement un service générique avec les 5 opérations de base déjà présentes dans le contrôleur d’animaux. (Donc oui c’est beaucoup du copié-collé des notes de cours, mais pour vos prochains projets vous allez pouvoir réutiliser, encore et encore, le service générique que vous vous apprêtez à créer !)
* Assurez-vous que vos 5 fonctionnalités (requêtes) fonctionnent encore bien en exécutant les deux projets.

**Étape 7 - L’arche de Noé**

* **54-56**  Créez un service spécialisé nommé **AnimalService**. Il héritera du service générique.
  + Remplacez l’injection du service générique dans **AnimalsController** par une injection d’**AnimalService**. (Après cette étape c’est promis, nous n’aurons plus l’air de faire des choses inutiles)
    - Cela devrait se résumer à modifier un type dans le constructeur et dans les variables de classe du contrôleur.
    - Tout est censé encore fonctionner. Testez pour être sûr, vous pourriez avoir oublié ou raté la configuration de l’injection du nouveau service dans Program.cs.
  + Dans **AnimalService**, codez une nouvelle méthode qui vide complètement le DbSet des animaux. (Donc qui supprime tous les animaux de la BD)
    - Inspirez vous de la méthode Delete dans le service générique.
    - Vous pouvez accéder au DbSet des animaux avec context.Animal.
    - La ligne de code suivante pourra vider le DbSet : \_context.Animal.RemoveRange(\_context.Animal);
  + Dans **AnimalsController**, créez une nouvelle action qui se servira de la méthode d’AnimalService permettant de vider les animaux de la BD.
    - Finalement, à partir du projet Angular, rendez le dernier bouton (« Buh-bye ») fonctionnel. Il doit appeler cette nouvelle action et permettre de vider la BD d’animaux.



Bon travail ! Gardez votre service générique à portée de main, vous pourrez le réutiliser dans tous vos projets d’ici la fin de la session.