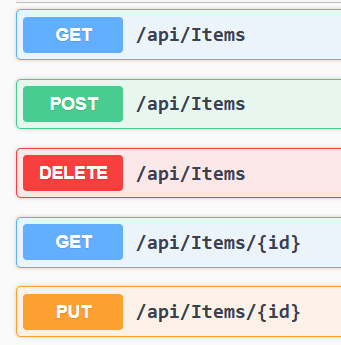
**Laboratoire #8**

Web API

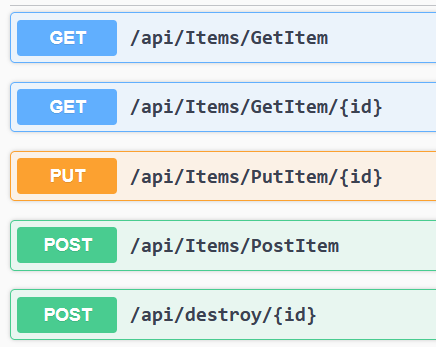
**Étape 1 - Mon premier projet Web Api 👶**

* ⭐ **8-10**  Créez un nouveau projet ASP.NET Core Web API avec le gabarit Web API. (Avec Core 8.0)
* 🗑 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 ITem.
  + Voici les propriétés d’un Item :
    - Id (int) Ex. 2
    - Name (string) Ex. "Silencieux d’autmobile"
    - Value (int) Ex. 80
* **13**  Installez les deux packages nécessaires.
* ⚙ 1**5-18**  Créez un contrôleur avec des actions auto-générées pour ce modèle. Cela devrait installer les dépendances Entity Framework.
  + Exécutez ensuite votre application et vérifiez que Swagger affiche les informations des requêtes de ce nouveau contrôleur. (L’ordre des requêtes pourrait être différent)



**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 code de l’action doit rester identique, cela servira encore à supprimer un Item 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**  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.

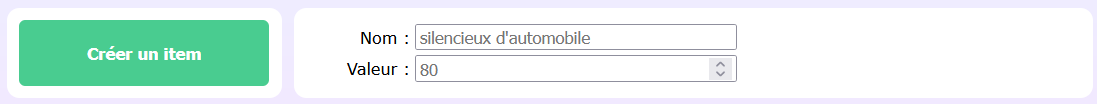
**Étape 5 - Angular ? Jamais entendu parlé**

* 📦 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 :
  + **31**  Créez un **Model** qui représente un **item**. (Calquez sur le Model d’item dans le projet **Web API**)
  + Décommentez deux variables de classe au composant App :
    - **items** : un **tableau** « d’items » possiblement **null** initialisé avec un tableau vide.
    - **item**: un « Item » possiblement **null**.
    - Dans le template HTML, décommentez la section « Affichage des items ».

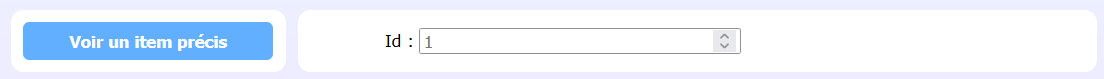
Pour les implémentations qui suivent, n’hésitez pas à jeter un coup d’œil à l’interface Swagger du Web API pour savoir quelles requêtes utiliser. Vous aurez à modifier un service et un composant.



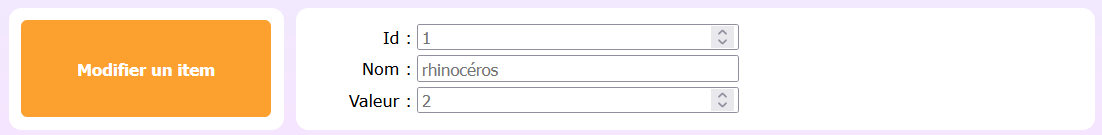
* Implémentation de **GET (All)**
  + **40-41**  Le but est de récupérer tous les items de la base de données, les stocker dans this.**items** et de les afficher.



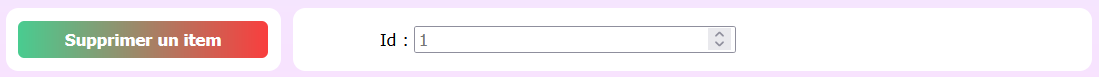
* Implémentation de **POST**
  + **38-39**  En utilisant la Value et le Name spécifiés par l’utilisateur, créez un new Item (**pas** dans la variable this.**item** !) qui sera envoyé avec la requête. Son Id doit ABSOLUMENT être 0. (Entity Framework va lui assigner un Id convenable lui-même)



* Implémentation de **GET**
  + **40-41**  En utilisant l’Id spécifié par l’utilisateur, remplissez la variable this.**item** avec l’item 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**
  + **42**  Créez un new Item (Pas dans la variable this.**item**) 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 item 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
  + 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’item 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

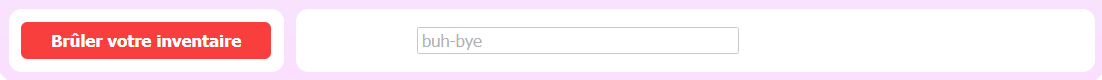
N’oubliez pas qu’il faut refaire la commande dotnet ef database update quand on change le projet de machine !

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

* **45-51**  Créez une classe nommée ItemsService et « glissez-la » entre le DbContext et le contrôleur d’items.
  + Le service se fera injecter le DbContext et le contrôleur se fera injecter le service.
  + On veut simplement un service avec les 5 opérations de base déjà présentes dans le contrôleur d’items. (Donc oui c’est beaucoup du copié-collé des notes de cours, mais adapté aux items)
* Assurez-vous que vos 5 fonctionnalités (requêtes) fonctionnent encore bien en exécutant les deux projets.
  + Le contrôleur ne doit plus avoir accès au DbContext ! (Vérifiez ses injections)

**Étape 7 - Inventaire aux poubelles**

* Pour cette dernière étape nous allons rendre fonctionnel le dernier bouton qui permet de supprimer tous les items.
  + Dans **ItemsService**, codez une nouvelle méthode qui vide complètement le DbSet des items. (Donc qui supprime tous les items de la BD)
    - Inspirez vous de la méthode Delete dans le service.
    - Vous pouvez accéder au DbSet des items avec \_context.Item.
    - La ligne de code suivante pourra vider le DbSet : \_context.Item.RemoveRange(\_context.Item);
  + Dans **ItemsController**, créez une nouvelle action qui se servira de la méthode d’ItemsService permettant de vider les items de la BD.
    - Finalement, à partir du projet Angular, rendez le dernier bouton (« Brûler votre inventaire ») fonctionnel. Il doit appeler cette nouvelle action et permettre de vider la BD d’items.



Bon travail !