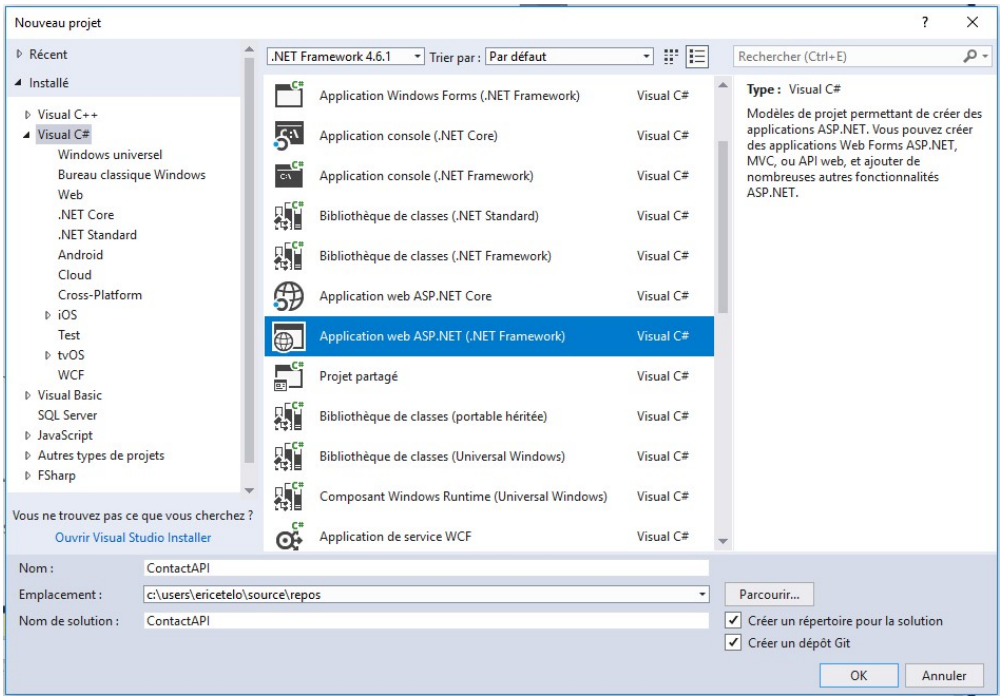
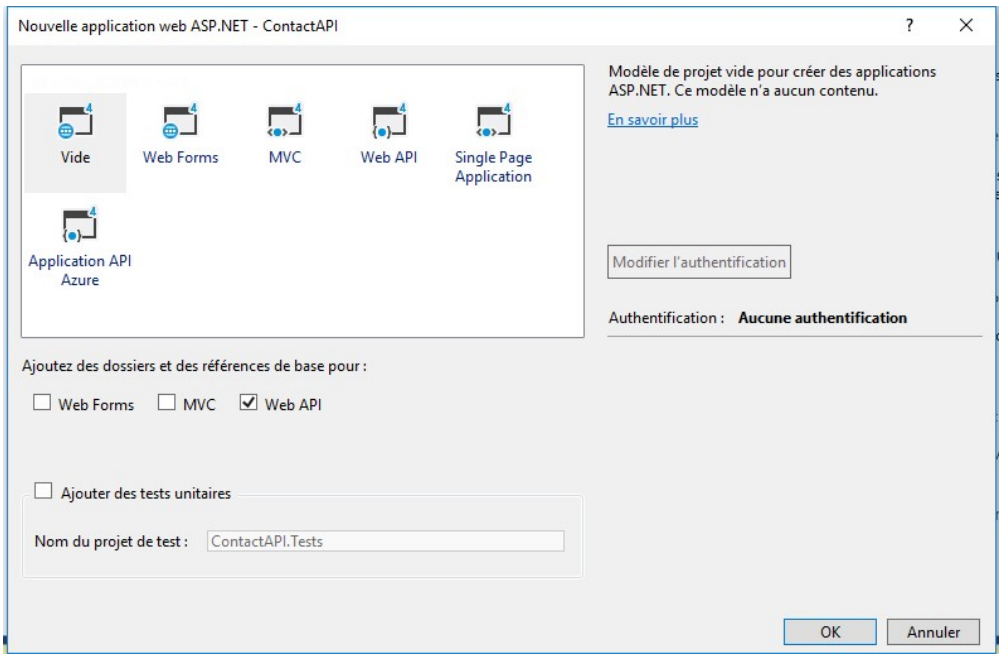
Création d'une API web ASP.NET RESTFULL (1)

LECTURE DE L'INTODUCTION ARCHITECTURE RESTFULL

1. Création du projet :

Créez un nouveau projet de type web ASP.NET. cochez web API et prenez un projet vide.





2. Création des classes :

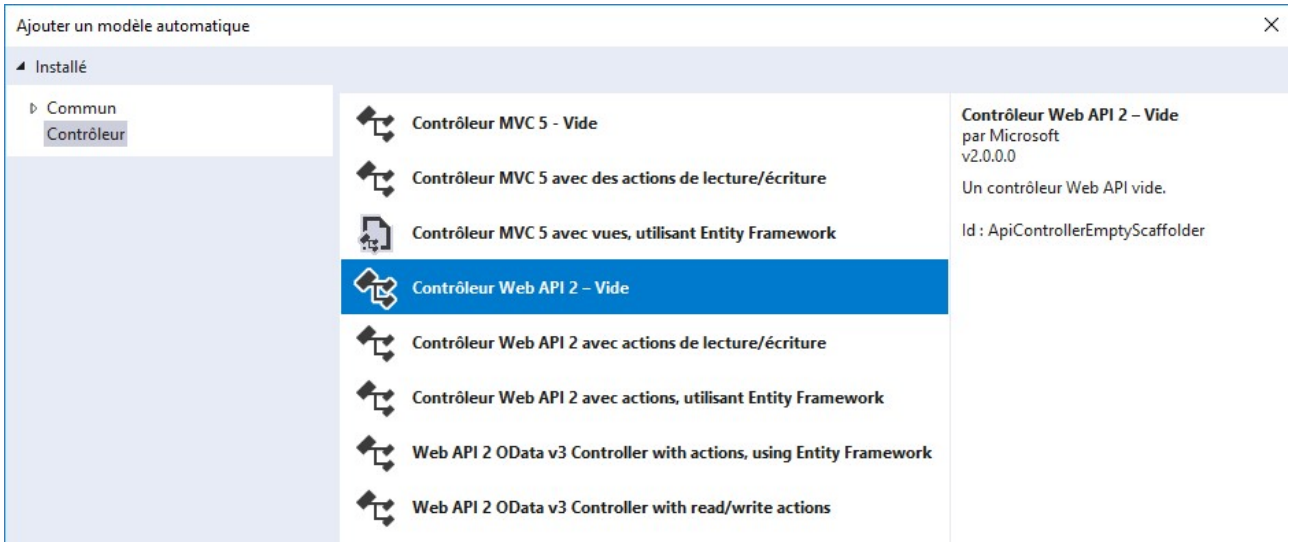
2.1. La classe Contact

Créez une nouvelle classe Contact dans le dossier modèle et placez-y le code suivant :

```
namespace ContactAPI.Models
{
    public class Contact
    {
        private int id;
        private string nom;
        private string prenom;
        private string email;
        private string numero;
        private double latitude;
        private double longitude;

        public int Id { get => id; set => id = value; }
        public string Nom { get => nom; set => nom = value; }
        public string Numero { get => numero; set => numero = value; }
        public string Prenom { get => prenom; set => prenom = value; }
        public string Email { get => email; set => email = value; }
        public double Latitude { get => latitude; set => latitude = value; }
        public double Longitude { get => longitude; set => longitude = value; }
    }
}
```

2.2. Le contrôleur

Créez une nouvelle classe contrôleur que vous nommerez ContactController



Le code de base sera le suivant :

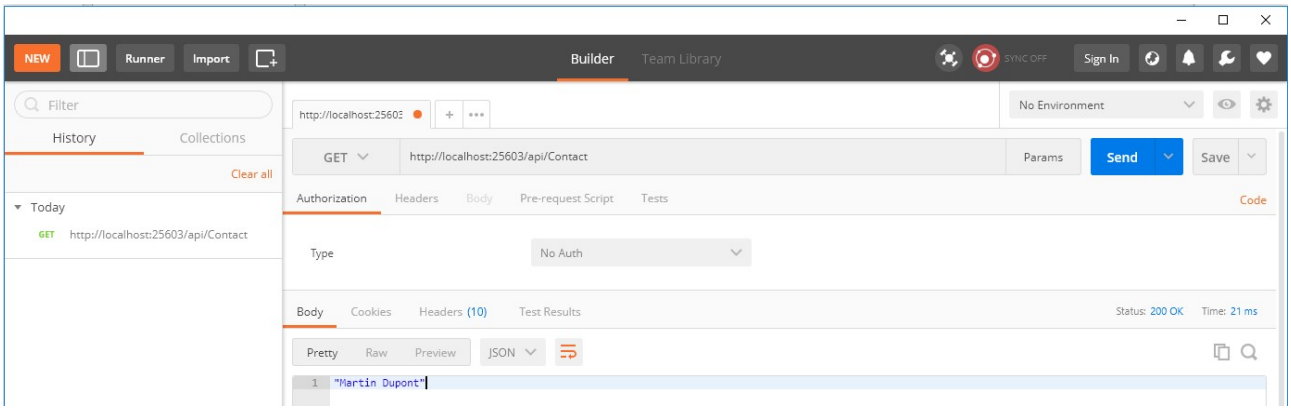
```
namespace ContactAPI.Controllers
{
    public class ContactController : ApiController
    {
        [HttpPost]
        public bool AddContact()
        {
            return true;
        }
        [HttpGet]
        public string GetContact()
        {
            return "Martin Dupont";

        }
        [HttpDelete]
        public string DeleteContact(string id)
        {
            return "Contact supprimé id " + id;

        }
        [HttpPut]
        public string UpdateContact(string Name, String Id)
        {
            return "Mise à jour du contact avec le nom " + Name + " and Id " + Id;

        }
    }
}
```

Essayez à ce stade de lancer l'exécution de votre projet. Une erreur devrait apparaître dans le navigateur. Ceci est normal car nous ne respectons pas la structure définie dans notre contrôleur, il n'existe pas de route par défaut sans rien spécifier comme l'indique le fichier WebApiConfig du dossier App_Start du projet. Modifiez l'url en y ajoutant /api/Contact

On observe ici que l'application nous renvoie bien une page avec Martin Dupont qui correspond à notre requête http GET définie dans le contrôleur.

3. Test des méthodes HTTP :

Lorsque nous appelons une route dans l'url du navigateur, celle-ci est appelée avec la méthode GET. Nous ne pouvons pas spécifier si nous voulons autre chose tel que du POST, PUT ou DELETE. Pour cela, il va falloir utiliser un outil spécifique qui nous permettra d'effectuer ce genre de requête. Le pus simple est d'utiliser un plugin intégré au navigateur. Pour chrome, vous pouvez installer Postman et lancer l'extension.



Dans cette fenêtre de Postman, testez les différentes requêtes que nous avons créées dans le contrôleur et observez les données retournées dans la réponse http.

A noter que pour les requêtes PUT, il faudra compléter le fichier WebApiConfig dont le schéma de route diffère de l'unique route actuellement définie :

```
namespace ContactAPI
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Configuration et services API Web

            // Itinéraires de l'API Web
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

            config.Routes.MapHttpRoute(
                name: "route1",
                routeTemplate: "api/{controller}/{name}/{id}",
                defaults: new {  }
            );
        }
    }
}
```

4. Implémentation de la persistance MySQL :

Ajoutez les librairies MySQL à votre projet d'API pour commencer. Créez ensuite une base de données avec une table « contact » qui reprendra pour champs les attributs définis dans la classe C#.

Créez ensuite une classe DAOContact qui assurera la persistance des contacts. Le code à la page suivante pourra vous inspirer. Nous avons un constructeur et une méthode pour récupérer tous nos contacts.

Vous créerez les méthodes manquantes qui permettront de faire les opérations CRUD élémentaires (CRUD : Create, Read, Update, Delete).

```
public class ContactDAO
    {
        private MySqlConnection conn;
        public ContactDAO()
        {
            string myConnectionString;
            myConnectionString = "server=192.168.1.13;uid=eric;pwd=btssio;database=contacts;";
            conn = new MySql.Data.MySqlClient.MySqlConnection();
            conn.ConnectionString = myConnectionString;
            conn.Open();
        }

        public List<Contact> getAllContacts()
        {
            List<Contact> lesContacts = new List<Contact>();
            string requete = "select * from contact";
            MySqlCommand cmd = new MySqlCommand(requete, conn);
            MySqlDataReader rdr = cmd.ExecuteReader();
            while (rdr.Read())
            {
                Contact c = new Contact(Convert.ToInt16(rdr[0].ToString()), rdr[1].ToString(),
rdr[2].ToString(), rdr[3].ToString(), rdr[4].ToString(), Convert.ToDouble(rdr[5].ToString()),
Convert.ToDouble(rdr[6].ToString()));
                lesContacts.Add(c);
            }
            rdr.Close();

            return lesContacts;
        }
    }
```

Nous allons ensuite compléter notre contrôleur avec le code de la page suivante. Nous avons complété ici certaines méthodes.

- HttpPost : nous remarquons l'utilisation d'un tag [FromBody] qui va nous permettre de récupérer depuis le corps de la requête http sous la forme d'un objet de type Contact. Le framework se chargera automatiquement de créer cet objet mais il faudra qu'un formulaire ait été soumis avec des noms de champs correspondant aux attributs de l'objet Contact.
- HttpGet paramétrée : Cette méthode renverra un objet dont l'identifiant sera passé dans l'url. C'est pour cela que par défaut, notre méthode renverra pour le moment par défaut l'élément 0 de la collection récupérée.
- HttpGet non paramétrée : ici, vu l'absence de paramètres, on renverra l'ensemble des objets présents dans la base de données. On pourrait créer d'autres méthodes avec d'autres paramètres selon le type d'information que l'on souhaite récupérer.

```
namespace ContactAPI.Controllers
{
    public class ContactController : ApiController
    {
        [HttpPost]
        public HttpResponseMessage AddContact([FromBody] Contact c)
        {
            if (c!= null)
                return Request.CreateResponse(HttpStatusCode.Created, c);
            else
                return Request.CreateResponse(HttpStatusCode.BadRequest, c);
        }
        [HttpGet]
        public Contact GetById(long id)
        {
            ContactDAO dao = new ContactDAO();
            List<Contact> lesContacts = dao.getAllContacts();
            return lesContacts.ElementAt(0);
        }
        [HttpGet]
        public IEnumerable<Contact> GetAll()
        {
            ContactDAO dao = new ContactDAO();
            List<Contact> lesContacts = dao.getAllContacts();
            return lesContacts.ToList();
        }

        [HttpDelete]
        public string DeleteContact(string id)
        {
            return "Contact supprimé id " + id;

        }
        [HttpPut]
        public string UpdateContact(string Name, String Id)
        {
            return "Mise à jour du contact avec le nom " + Name + " and Id " + Id;

        }
    }
```

Testez ces méthodes avec les appels REST adaptés dans PostMan avec les différents paramètres possibles.

Vous compléterez les méthodes manquantes pour avoir une API totalement fonctionnelle.

Quelques liens complémentaires :

http://www.c-sharpcorner.com/UploadFile/0c1bb2/creating-Asp-Net-web-api-rest-service/
https://james.boelen.ca/programming/webapi-routes-optional-parameters-constraints/