

Rapport PPE 1 – Service Web

Jacques Bourbonnais

Sommaire

BTS Services informatiques aux organisations	2
Définition du besoin en respectant le cahier de charges	4
Conception de la BDD : MCD, MLD	4
Construction de la BDD	5
Peuplement de la BDD	6
Création d'un projet Web Api avec Visual Studio 2022.....	7
Création d'une couche d'accès aux données (ORM Object Relationnal Mapping) basé sur les Entity Framework.....	8
Création des objets model	8
Création des contrôleurs (avec les verbes GET, POST, PUT, DELETE).....	9
Sécurisation des api avec un JWT Json Web Token	10
Test des api via swagger ou talent Api	13
Publication GitHub	15

DESCRIPTION D'UNE RÉALISATION PROFESSIONNELLE		N° réalisation : 01
Nom, prénom : Bourbonnais Jacques		N° candidat : 02345657150
Épreuve ponctuelle <input checked="" type="checkbox"/>	Contrôle en cours de formation <input type="checkbox"/>	Date : 13 / 06 / 2024
Organisation support de la réalisation professionnelle Laboratoire GSB		
Intitulé de la réalisation professionnelle PROJET Service Web		
Période de réalisation : 03/24 Lieu : EPSI Lyon		
Modalité : <input checked="" type="checkbox"/> Seul(e) <input type="checkbox"/> En équipe		
Compétences travaillées <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Concevoir et développer une solution applicative <input checked="" type="checkbox"/> Assurer la maintenance corrective ou évolutive d'une solution applicative <input checked="" type="checkbox"/> Gérer les données 		
Conditions de réalisation¹ (ressources fournies, résultats attendus) <ul style="list-style-type: none"> - Ressources fournies : description du contexte, expression des besoins, exemple de requête - Résultats attendus : création, conception et remplissage d'une base de données - développement d'une application Back-end appelant la base de données en utilisant la norme RESTful - sécurisation des API via un token JWT 		
Description des ressources documentaires, matérielles et logicielles utilisées² <ul style="list-style-type: none"> - Modélisation : Looping - SGBD : SQL Server et SQL Management Studio - Environnement de développement : Visual Studio - Bibliothèque de développement : .NET, Entity Framework (ORM) - Langages : SQL, C#, TransactSQL - Gestion de version : Github - Tests des comportement API : Talend API Tester 		
Modalités d'accès aux productions et à leur documentation³ <ul style="list-style-type: none"> - GitHub 		

¹ En référence aux *conditions de réalisation et ressources nécessaires* du bloc « Conception et développement d'applications » prévues dans le référentiel de certification du BTS SIO.

² Les réalisations professionnelles sont élaborées dans un environnement technologique conforme à l'annexe II.E du référentiel du BTS SIO.

³ Lien vers la documentation complète, précisant et décrivant, si cela n'a été fait au verso de la fiche, la réalisation professionnelle, par exemples service fourni par la réalisation, interfaces utilisateurs, description des classes ou de la base de données.

Descriptif de la réalisation professionnelle, y compris les productions réalisées et schémas explicatifs

Développement d'un Service Web (Contexte GSB) : il s'agit de développer un back-end respectant les concepts de l'architecture RESTful

- Définition du besoin en respectant le cahier de charges
- Conception de la BDD : MCD, MLD et MPD
- Construction de la BDD
- Peuplement de la BDD à partir d'un jeu de données exporté depuis l'interface OpenDate du ministère de l'intérieur pour les données des Départements
- Peuplement de la Spécialité à partir du site de la CPAM
- Création d'un projet WebApi avec Visual Studio 2022
- Création d'une couche d'accès aux données (ORM Object Relational Mapping) basé sur les Entity Framework
- Création des objets model
- Création des contrôleurs (avec les verbes GET, POST, PUT, DELETE)
- Sécurisation des api avec un JWT Json Web Token
- Mise en place du HTTPS via un certificat auto signé
- Exécution via IIS (serveur web intégré à Visual Studio)
- Test des api via swagger ou talent Api
- Gestion de versions
- Publication GitHub

Définition du besoin en respectant le cahier de charges

Le laboratoire Galaxy Swiss Bourdin (GSB) est amené dans ses différentes activités à contacter les médecins installés, par exemple les visiteurs médicaux du laboratoire se déplacent afin de présenter les nouveaux produits pharmaceutiques.

Une base de données de médecins est utilisée dans ces différents processus. Cette base de données évolue fréquemment, ceci à cause de départ à la retraite de médecins ou d'installation de nouveaux praticiens. Les applications doivent intégrer ces évolutions des données.

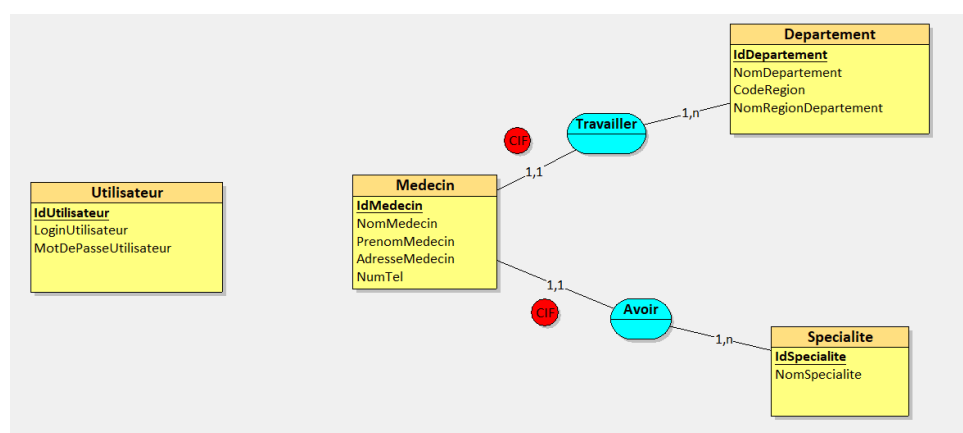
Il a été décidé de confier à une société de services informatiques la responsabilité de développer un service web donnant accès aux informations de la base de données.

Le but de l'application est de développer un Web Service dans un contexte back-end en respectant les concepts de l'architecture RESTful.

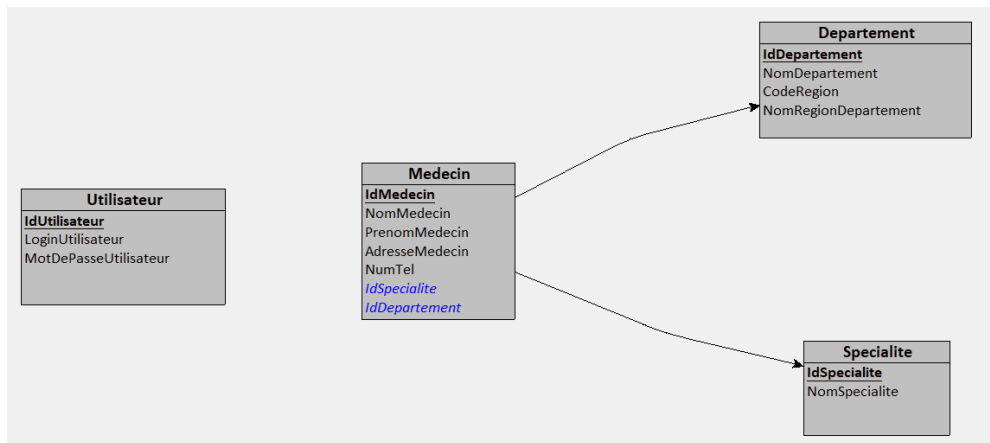
Pour les besoins du développement, nous aurons plusieurs éléments essentiels dans un premier temps. Tout d'abord, nous aurons besoin d'un environnement de développement intégré (IDE). Dans notre cas, j'ai opté pour Visual Studio 2022. Ensuite, nous aurons besoin d'un environnement pour la gestion de la base de données. J'ai pris la décision de rester sur les services Microsoft pour mener à bien ce projet, et par conséquent, j'ai choisi SQL Server Management Studio 2019. De plus, j'ai dû conceptualiser la base de données, et j'ai utilisé le logiciel Looping à cette fin.

Conception de la BDD : MCD, MLD

Modèle Conceptuel de Données : En utilisant l'outil Looping, voici le MCD que j'ai réalisé :



Modèle Logique de Données : Suite au MCD, voici le MLD :

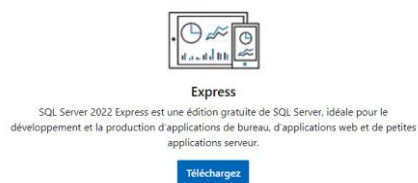


Construction de la BDD

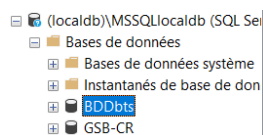
La base de données se fera sur SQL Server Management Studio (SSMS).

Pour installer SSMS :

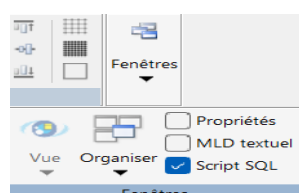
<https://learn.microsoft.com/fr-fr/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>



Ensuite, il faut créer une base de données.



Puis faire une nouvelle requête (ou faire ctrl + n). Et dans la requête, il faut saisir le code MySQL pour construire la base de données. Pour avoir tout le script de la base de données, il faut aller sur Looping (où on a fait le MCD et le MLD), et cliquer sur fenêtre en haut à droite et cocher la case « Script SQL ».



Et voici le code MySQL généré par Looping.

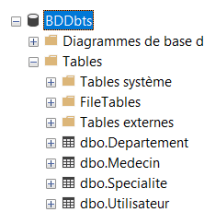
```
SQL
CREATE TABLE Utilisateur(
  IdUtilisateur INT IDENTITY,
  LoginUtilisateur VARCHAR(50) NOT NULL,
  MotDePasseUtilisateur VARCHAR(50) NOT NULL,
  PRIMARY KEY(IdUtilisateur)
);

CREATE TABLE Specialite(
  IdSpecialite INT IDENTITY,
  NomSpecialite VARCHAR(50) NOT NULL,
  PRIMARY KEY(IdSpecialite)
);

CREATE TABLE Departement(
  IdDepartement VARCHAR(3),
  NomDepartement VARCHAR(50) NOT NULL,
  CodeRegion VARCHAR(3) NOT NULL,
  NomRegionDepartement VARCHAR(50) NOT NULL,
  PRIMARY KEY(IdDepartement)
);

CREATE TABLE Medecin(
  IdMedecin INT IDENTITY,
  NomMedecin VARCHAR(50) NOT NULL,
  PrenomMedecin VARCHAR(50) NOT NULL,
  AdresseMedecin VARCHAR(50) NOT NULL,
  NumTel VARCHAR(10) NOT NULL,
  IdSpecialite INT NOT NULL,
  IdDepartement VARCHAR(3) NOT NULL,
  PRIMARY KEY(IdMedecin),
  FOREIGN KEY(IdSpecialite) REFERENCES Specialite(IdSpecialite),
  FOREIGN KEY(IdDepartement) REFERENCES Departement(IdDepartement)
);
```

Ensuite, il faut copier tout le script et le coller dans la nouvelle requête, et exécuter le code. En faisant ceci, notre base de données va se remplir de 4 tables (Utilisateur, Specialite, Departement et Medecin).



Peuplement de la BDD

Passons maintenant au peuplement de la base de données. Pour remplir la table des départements nous avons utilisé des données provenant du site Open Data du ministère de l'intérieur via un fichier Excel .CSV

	A	B	C	D	E	F
1	IdDepartement	NomDepartement	CodeRegion	NomRegion	Departement	
2	1	Ain	84	Auvergne-Rhône-Alpes		
3	2	Aisne	32	Hauts-de-France		
4	3	Allier	84	Auvergne-Rhône-Alpes		
5	4	Alpes-de-Haute-Provence	93	Provence-Alpes-Côte d'Azur		
6	5	Hautes-Alpes	93	Provence-Alpes-Côte d'Azur		
7	6	Alpes-Maritimes	93	Provence-Alpes-Côte d'Azur		
8	7	Ardèche	84	Auvergne-Rhône-Alpes		
9	8	Ardennes	44	Grand Est		
10	9	Ariège	76	Occitanie		
11	10	Aube	44	Grand Est		
12	11	Aude	76	Occitanie		
13	12	Aveyron	76	Occitanie		
14	13	Bouches-du-Rhône	93	Provence-Alpes-Côte d'Azur		
15	14	Calvados	28	Normandie		
16	15	Cantal	84	Auvergne-Rhône-Alpes		
17	16	Charente	75	Nouvelle-Aquitaine		
18	17	Charente-Maritime	75	Nouvelle-Aquitaine		
19	18	Cher	24	Centre-Val de Loire		
20	19	Corrèze	75	Nouvelle-Aquitaine		
21	21	Côte-d'Or	27	Bourgogne-Franche-Comté		
22	22	Côtes-d'Armor	53	Bretagne		
23	23	Creuse	75	Nouvelle-Aquitaine		
24	24	Dordogne	75	Nouvelle-Aquitaine		
25	25	Doubs	27	Bourgogne-Franche-Comté		
26	26	Drôme	84	Auvergne-Rhône-Alpes		
27	27	Eure	28	Normandie		

Il faut bien faire attention de mettre le bon nom des Primary Keys dans la ligne une.
Et pour les 3 autres tables, je les ai remplis moi-même en utilisant du MySQL.

La table Utilisateur :

	IdUtilisateur	LoginUtilisateur	MotDePasseUtilisateur
1	27	Jacques	Jacc123
2	28	Hugo	Hugo123
3	30	Bouh	Bouh123
4	31	Abdulrachid	Abdul123
5	32	Damarice	Dama123
6	33	Arthur	voiture123!

La table Specialite :

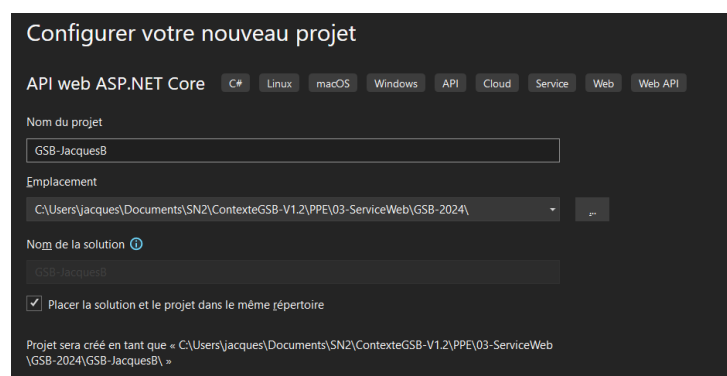
	IdSpecialite	NomSpecialite
1	30	Dermatologie et vénérérologie
2	31	Ophthalmologie
3	32	Néphrologie
4	33	Médecine interne
5	34	Cardiologie et maladies vasculaires
6	35	Radiodiagnostic et imagerie médicale
7	36	Neurologie
8	37	O.R.L et chirurgie cervico-faciale
9	38	Oncologie
10	39	Gastro-entérologie et hépatologie
11	40	Anesthésie-réanimation
12	41	Rhumatologie
13	42	Neurochirurgie
14	43	Hématologie
15	44	Médecin nucléaire
16	45	Pneumologie
17	46	Chirurgie générale

La table Medecin :

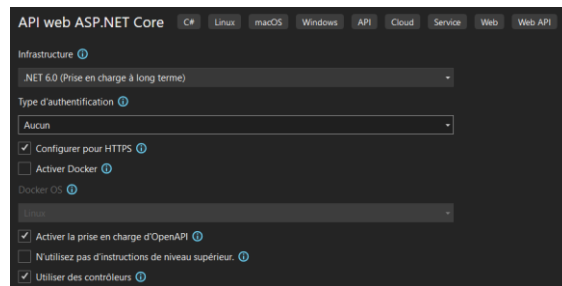
	IdMedecin	NomMedecin	PrenomMedecin	AdresseMedecin	NumTel	IdSpecialite	IdDepartement
1	13	CIVEL	Saul	Avenue des Champs-Élysées	0683749274	31	26
2	14	ONETO	Hugo	Rue de la Liberté	0675937502	38	69
3	15	COUTURIER	Malo	Place de la Bastille	0621926402	43	75
4	16	LAUTREDOUX	Luca	Rue du Faubourg Saint-Antoine	0649143093	48	974
5	17	BERJAOUI	Alexandre	Boulevard Haussmann	0640325688	55	7

Création d'un projet Web Api avec Visual Studio 2022

Sur Visual Studio 2022, il faut sélectionner un projet API web ASP.NET Core.

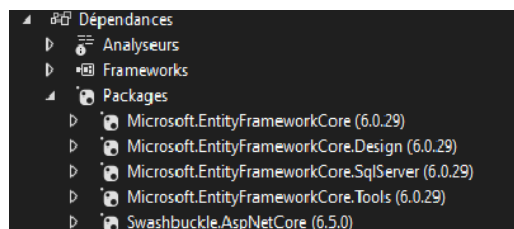


Lors des paramètres, il faut bien mettre la version .NET 6.0 pour l'installation des paquets NuGets. Et aussi configurer le certificat HTTPS (version sécurisée du protocole HTTP) en cochant la case.



Création d'une couche d'accès aux données (ORM Object Relationnal Mapping) basé sur les Entity Framework

Pour cette partie, il faut ajouter les packages NuGet, voici un des packages qu'il faut installer. (Comme nous avons la version .NET 6.0 il faut télécharger la dernière version 6.0 des différents packages). Voici les différents packages qu'on aura installé.



Création des objets model

L'installation des packages nous permet donc de créer les différents objets model class. Pour cela, il faut ouvrir la Console du Gestionnaire de package dans l'onglet « Outils » de Visual Studio 2022.

Puis il faut insérer cette commande :

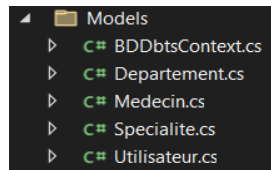
```
scaffold-DbContext "Data Source=(localdb)\mssqllocaldb;Initial Catalog=NomBDD;Integrated Security=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models;
```

Pour le **NomBDD**, je dois mettre la mienne donc « BDDbts », donc dans la console pour ma part, je dois mettre :

```
scaffold-DbContext "Data Source=(localdb)\mssqllocaldb;Initial Catalog=BDDbts;Integrated Security=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models;
```

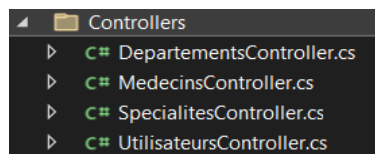

Et en exécutant cette commande :

Si c'est écrit « Build succeeded », cela veut dire que tous vos objets class model de la base de données se sont créés.

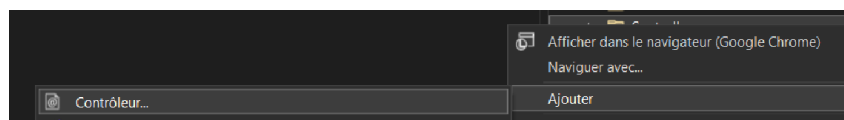


Création des contrôleurs (avec les verbes GET, POST, PUT, DELETE)

J'ai créé 4 contrôleurs Departements, Medecins, Specialites et Utilisateurs.



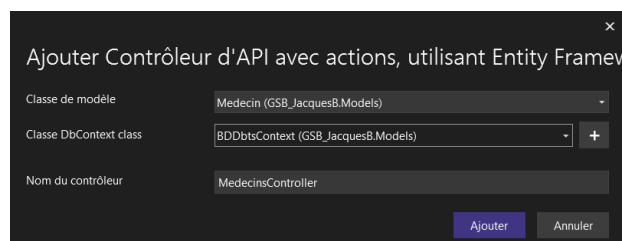
Je vais montrer la création d'un seul contrôleur parce que c'est répétitif et il faut faire la même chose pour les 4. Il faut d'abord faire un clic droit sur le dossier « Controllers » et Ajouter.



Puis il faut sélectionner contrôleur d'API.

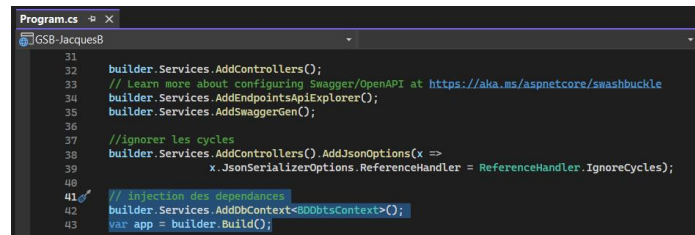


Ensuite sélectionner la classe de modèle Medecin et le DbContext qui est pour moi BDDbts.



Il faut refaire ceci 3 fois mais pas avec la classe Medecin mais avec les classes Departement, Specialite et Utilisateur.

Ensuite, afin de pouvoir effectuer des opérations telles que la lecture, l'écriture et la modification de données, il faut que le Service Web (l'application back-end) puisse interagir avec la base de données, donc pour cela, il faut injecter les dépendances dans le contexte du modèle BDDbts.

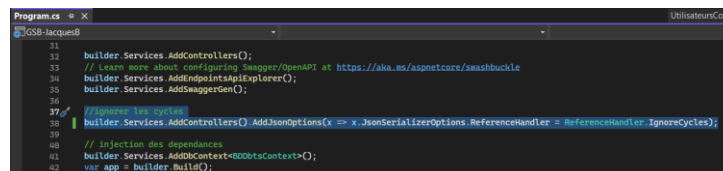


```
31
32 builder.Services.AddControllers();
33 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
34 builder.Services.AddEndpointsApiExplorer();
35 builder.Services.AddSwaggerGen();
36
37 // Ignorer les cycles
38 builder.Services.AddControllers().AddJsonOptions(x =>
39     x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);
40
41 // Injection des dépendances
42 builder.Services.AddDbContext<BDDbtsContext>();
43 var app = builder.Build();
```

`Builder.Services.AddDbContext<BDDbtsContext>();`

Cette ligne qui est ajoutée dans le fichier « program.cs » nous permet, par exemple, de pouvoir ajouter un nouveau médecin (POST), d'accéder à la liste de médecins (GET), de modifier un médecin, de mettre les détails d'un médecin à jour (PUT) et de supprimer un médecin (DELETE). (Hiérarchie RESTful).

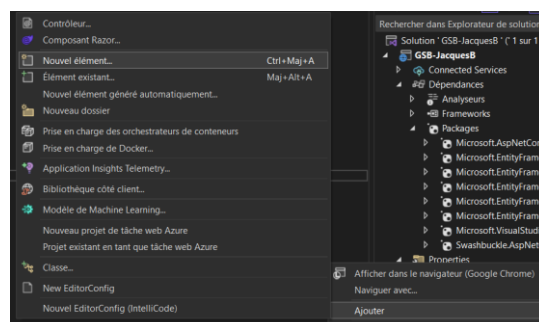
Ensuite, nous incluons l'option de passer outre les cycles détectés lors de la sérialisation. Dans le même fichier « program.cs ».



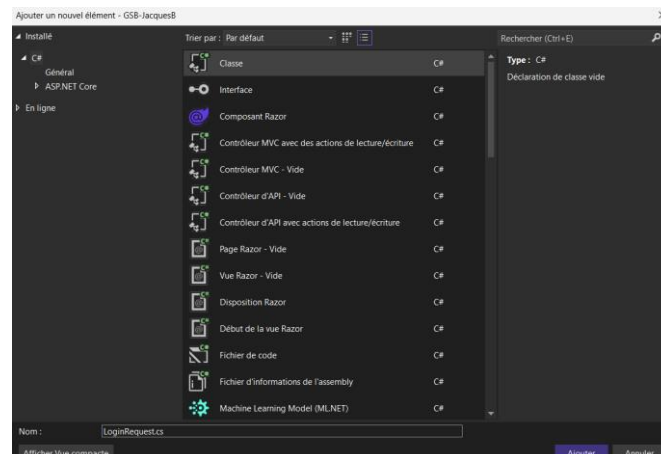
```
31
32 builder.Services.AddControllers();
33 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
34 builder.Services.AddEndpointsApiExplorer();
35 builder.Services.AddSwaggerGen();
36
37 // Ignorer les cycles
38 builder.Services.AddControllers().AddJsonOptions(x => x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);
39
40 // Injection des dépendances
41 builder.Services.AddDbContext<BDDbtsContext>();
42 var app = builder.Build();
```

Sécurisation des api avec un JWT Json Web Token

Passons maintenant à la sécurisation de l'API pour éviter qu'une personne puisse supprimer ou faire quoique ce soit d'autre dans notre base de données, on va d'abord commencer par créer un nouveau modèle.



Après avoir cliqué sur « Nouvel élément », il faudra créer une classe qu'on nommera LoginRequest.cs



Une fois notre classe créée, nous allons y rajouter deux variables, une pour le nom d'utilisateur et une pour le mot de passe.

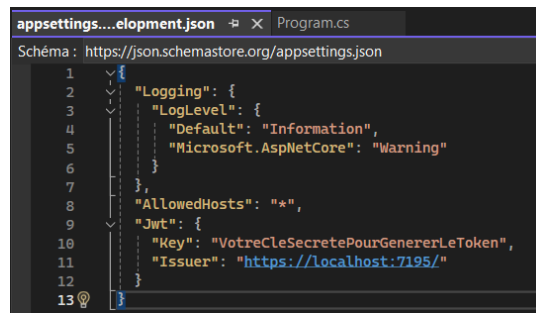
```
1 namespace GSB_JacquesB.Models
2 {
3     1 référence
4     public class LoginRequest
5     {
6         1 référence
7         public string? username { get; set; }
8         1 référence
9         public string? password { get; set; }
10    }
11 }
```

La classe modèle est créée, on peut donc passer à la création du contrôleur qu'on va nommer « TokenController.cs ». Et on va y ajouter un script pour la sécurisation par token.

```
TokenController.cs  appsettings.Development.json  Program.cs
GSB-JacquesB
GSB_JacquesB.Controllers.TokenController

1 using GSB_JacquesB.Models;
2 using Microsoft.AspNetCore.Mvc;
3 using Microsoft.AspNetCore.Mvc.Core;
4 using Microsoft.IdentityModel.Tokens;
5 using System.IdentityModel.Tokens.Jwt;
6 using System.Text;
7
8 namespace GSB_JacquesB.Controllers
9 {
10    [Route("api/[controller]")]
11    [ApiController]
12    public class TokenController : ControllerBase
13    {
14        private readonly ApplicationDbContext _context;
15
16        //public TokenController(ApplicationDbContext context)
17        //{
18            // _context = context;
19        //}
20
21        private IConfiguration _config;
22
23        public TokenController(IConfiguration config, ApplicationDbContext context)
24        {
25            _config = config;
26            _context = context;
27        }
28
29        [HttpPost]
30        public IActionResult Post([FromBody] LoginRequest loginRequest)
31        {
32            //your logic for login process
33            //if login username and password are correct then proceed to generate token
34
35            Utilisateur? user = _context.Utilisateurs.Where(u => u.LoginUtilisateur.Equals(loginRequest.username) && u.MotDePasseUtilisateur.Equals(loginRequest.password)).FirstOrDefault();
36
37            if (user != null)
38            {
39                var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
40                var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
41
42                var token = new JwtSecurityToken(_config["Jwt:Issuer"],
43                    _config["Jwt:Audience"],
44                    null,
45                    expires: DateTime.Now.AddMinutes(120),
46                    signingCredentials: credentials);
47
48                var tokenHandler = new JwtSecurityTokenHandler();
49                var token = tokenHandler.WriteToken(token);
50
51                return Ok(token);
52            }
53            return BadRequest();
54        }
55    }
56 }
```

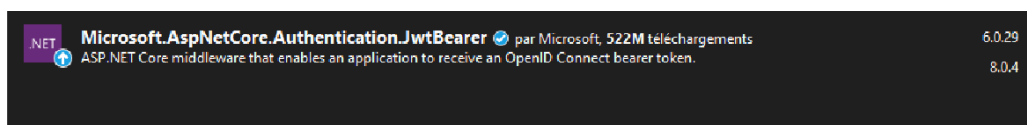
On va ensuite se rendre dans le fichier « appsettings.Development.json » pour y ajouter les informations de connexion du token, pour générer notre clé hachée de notre token.



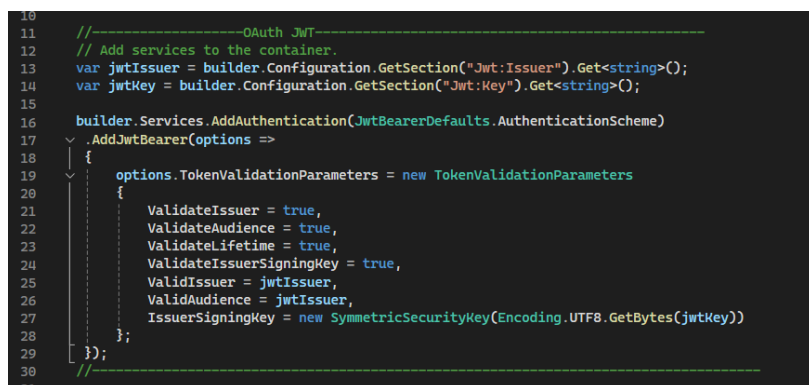
The screenshot shows the 'appsettings.Development.json' file in Visual Studio. The JSON content is as follows:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Jwt": {
    "Key": "VotreCleSecretePourGenererLeToken",
    "Issuer": "https://localhost:7195/"
  }
}
```

Pour finaliser l'authentification par token il nous de rajouter un package NuGet.



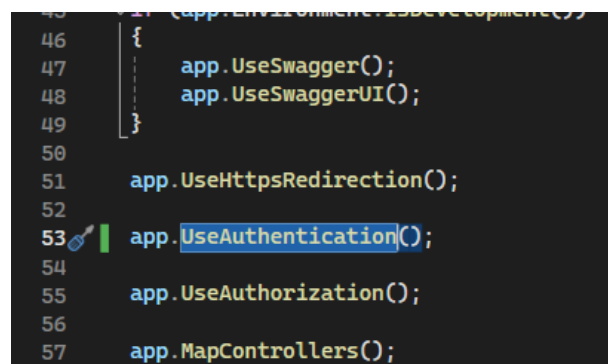
Rajouter ces lignes de codes dans le fichier « program.cs ».



The screenshot shows the 'program.cs' file in Visual Studio. The code is as follows:

```
10 //-----OAuth JWT-----
11 // Add services to the container.
12 var jwtIssuer = builder.Configuration.GetSection("Jwt:Issuer").Get<string>();
13 var jwtKey = builder.Configuration.GetSection("Jwt:Key").Get<string>();
14
15 builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
16 .AddJwtBearer(options =>
17 {
18     options.TokenValidationParameters = new TokenValidationParameters
19     {
20         ValidateIssuer = true,
21         ValidateAudience = true,
22         ValidateLifetime = true,
23         ValidateIssuerSigningKey = true,
24         ValidIssuer = jwtIssuer,
25         ValidAudience = jwtIssuer,
26         IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey))
27     };
28 });
29
30
31
```

L'application utilise donc un système d'authentification donc il faut rajouter dans le même fichier « app.UseAuthentication() ».



The screenshot shows the 'app.UseAuthentication()' method call in the 'program.cs' file. The code is as follows:

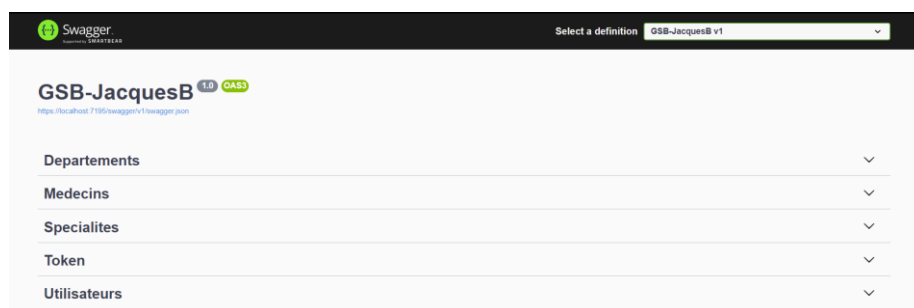
```
45
46 {
47     app.UseSwagger();
48     app.UseSwaggerUI();
49 }
50
51 app.UseHttpsRedirection();
52
53 app.UseAuthentication();
54
55 app.UseAuthorization();
56
57 app.MapControllers();
```

Et la dernière étape, pour mettre en place cette utilisation de token pour sécuriser les tables de la base de données, par exemple pour sécuriser la table Medecin de la base de données, je vais mettre dans le contrôleur Medecins « [Authorize] ».

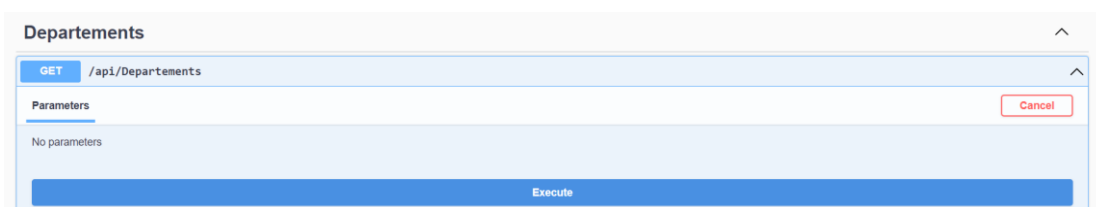
```
namespace GSB_JacquesB.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    1 référence
    public class MedecinsController : ControllerBase
    {
    }
```

Test des api via swagger ou talent Api

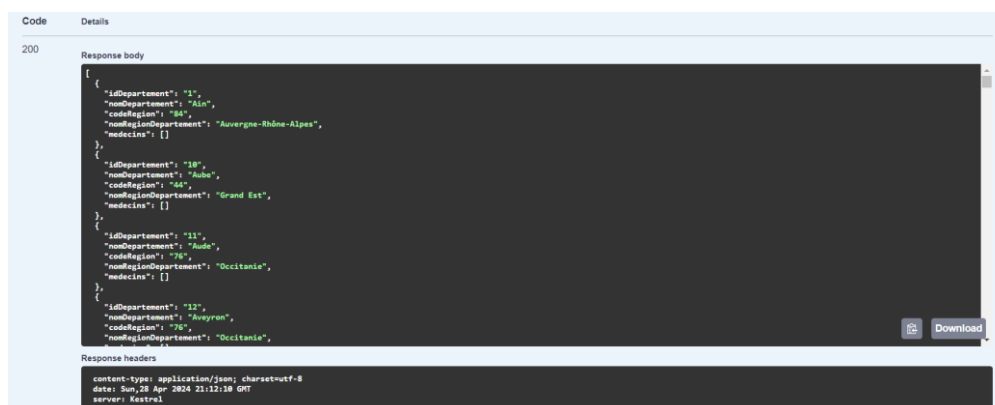
Maintenant, voici le lancement de l'application pour montrer son bon fonctionnement.



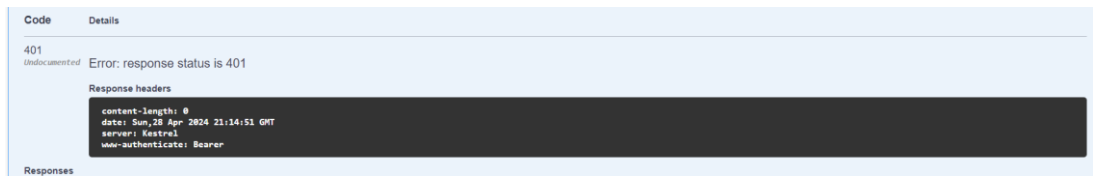
Testons d'abord d'afficher la liste des départements qui n'est pas sécurisé par un token. Il faut appuyer sur le compartiment « GET », puis « Try it out », puis « Execute ».



On voit qu'on a pu accéder aux informations de la base de données sans problèmes.

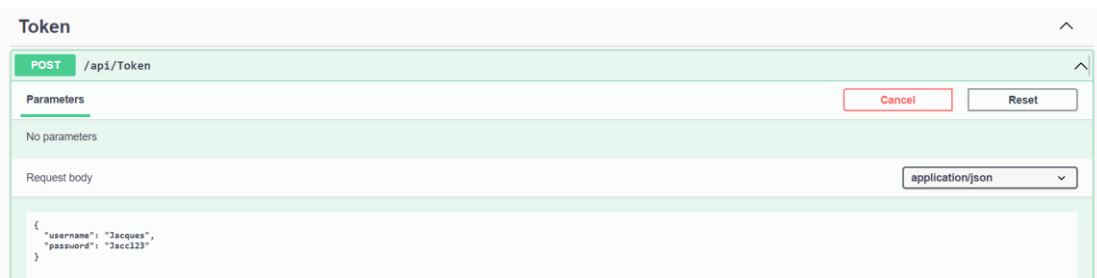


Maintenant regardons si nous pouvons voir la liste des médecins.



Une erreur survient car il y a une authentification Bearer qu'empêche l'accès aux données.

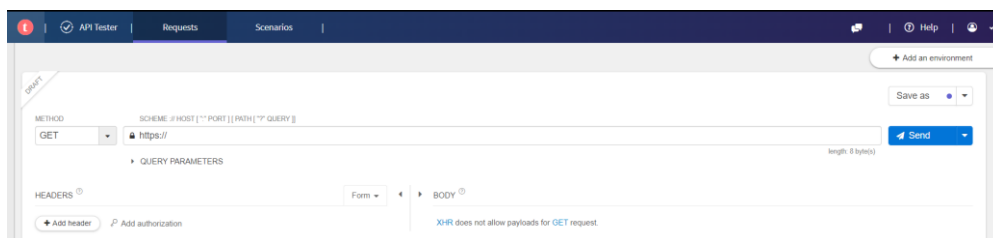
Pour avoir accès à la liste de médecins et pour gérer la table Medecin entièrement. Il faut d'abord prendre le token en s'identifiant avec un username et password (table Utilisateur).



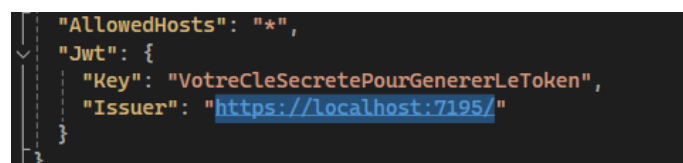
Le username et password appartiennent bien à la base de données donc je vais avoir accès à la clé du token.



Une fois la clé récupérée, nous devons utiliser Talend API Tester car Swagger ne nous permet pas d'avoir accès à la liste de médecins avec le token, il nous donne simplement le token.





Une fois sur cette interface, on laisse la méthode GET, et pour l'url https on met le localhost qu'on retrouve dans notre fichier « appsettings.Development.json ».



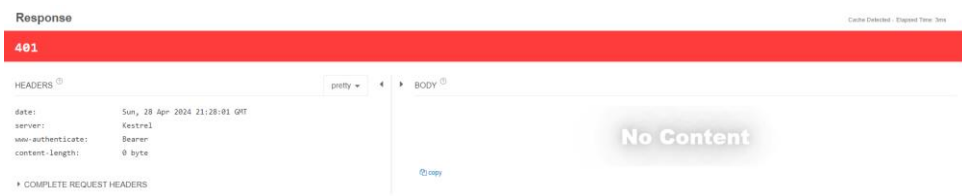
Et on rajoute /api/Medecins.

METHOD SCHEME // HOST [":" PORT] [PATH ["?" QUERY]]




GET 

 https://localhost:7195/api/Medecins

On peut essayer mais cela ne fonctionnera toujours pas car il faut qu'on inclue le token.

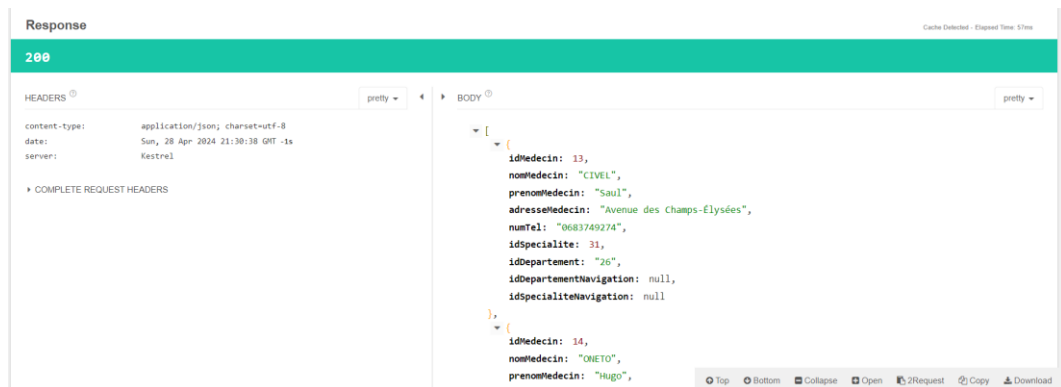


Donc pour qu'on puisse avoir accès à la liste, il faut rajouter un header et mettre Bearer + token.

HEADERS   Form 

☒ Authorization : Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC...

Et si on envoie une requête, on remarque qu'on a récupéré la liste de médecins.



Publication GitHub

Pour déposer le projet sur GitHub, nous devons d'abord commencer par installer Git.



Ensuite, il faut se rendre sur notre compte GitHub et créer un nouveau repository dans lequel nous allons déposer le projet.

Ensuite il faut faire un clic droit sur le dossier contenant tous les fichiers du projet.

Et sélectionner « Afficher plus d'options » → « Ouvrir Git Bash ».

Puis une fenêtre va s'ouvrir et il faudra initialiser le dossier comme étant un dossier git en tapant la commande « git init ».

```
jacques@MSI MINGW64 ~/Documents/SN2/ContexteGSB-V1.2/PPE/03-ServiceWeb
$ git init
Initialized empty Git repository in C:/Users/jacques/Documents/SN2/ContexteGSB-V1.2/PPE/03-ServiceWeb/.git/
```

Puis il faut se connecter son dépôt local avec le dépôt GitHub.

```
jacques@MSI MINGW64 ~/Documents/SN2/ContexteGSB-V1.2/PPE/03-ServiceWeb (master)
$ git remote add origin https://github.com/AzzJac/GSB-GestionComptesRendus.git
```

Ensuite, il faut taper la commande « git add -A », ceci permet d'ajouter tous les fichiers dans la zone de préparation, une étape intermédiaire entre les modifications et le prochain commit.

On va effectuer notre premier commit, pour cela, il faut écrire la commande « git commit -m [commentaire du commit] ».

```
jacques@MSI MINGW64 ~/Documents/SN2/ContexteGSB-V1.2/PPE/03-ServiceWeb (master)
$ git commit -m "first push"
[master (root-commit) 943be22] first push
326 files changed, 19013 insertions(+)
create mode 100644 Dossier PPE Service Web.docx
create mode 100644 GSB-2024/GSB-JacquesB/.vs/GSB-JacquesB/DesignTimeBuild/.dtbc
```

Une fois le commit fait, on a plus qu'à envoyer les fichiers sur GitHub en faisant un push.

```
jacques@MSI MINGW64 ~/Documents/SN2/ContexteGSB-V1.2/PPE/03-ServiceWeb (master)
$ git push origin master
Enumerating objects: 435, done.
Counting objects: 100% (435/435), done.
Delta compression using up to 20 threads
Compressing objects: 100% (390/390), done.
Writing objects: 100% (435/435), 30.44 MiB | 6.59 MiB/s, done.
Total 435 (delta 126), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (126/126), done.
To https://github.com/AzzJac/GSB-GestionComptesRendus.git
* [new branch]      master -> master
```


Une fois terminé, on peut aller vérifier que le projet existe sur GitHub.

