

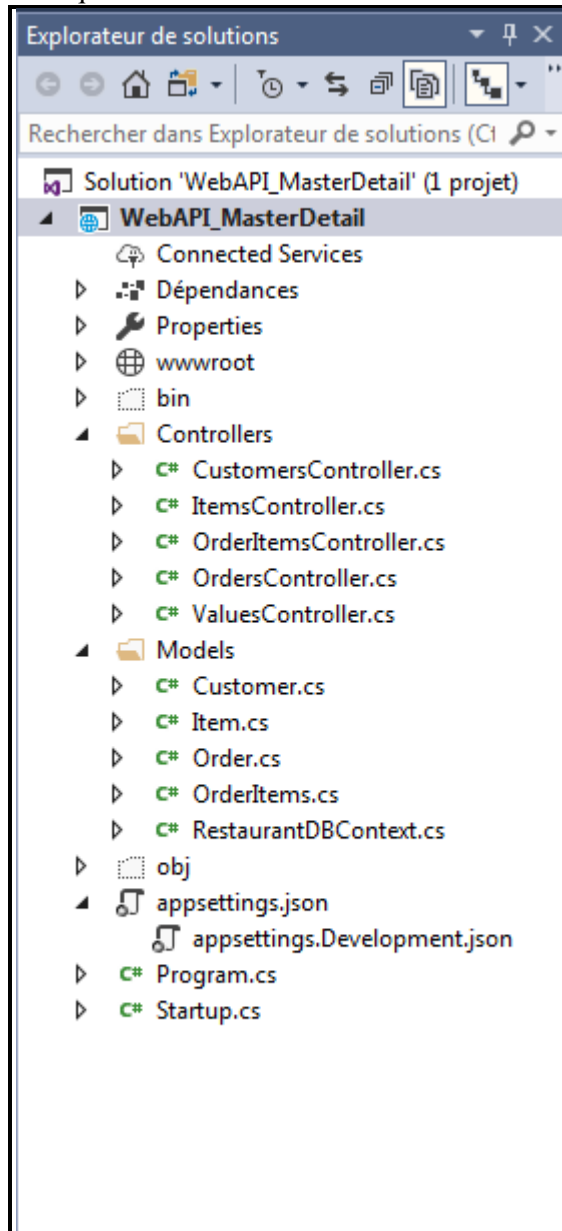
## JWT token authentication in ASP.NET Core 2.1 with Visual Studio 2017

Nous allons aborder dans ce tutoriel la notion de *JWT Authentication* dans ASP.NET Core 2.1.

Dans l'application web ASP.NET Core qu'on a nommée RestaurantApp, on va la garnir par pas mal de mesures de sécurité.

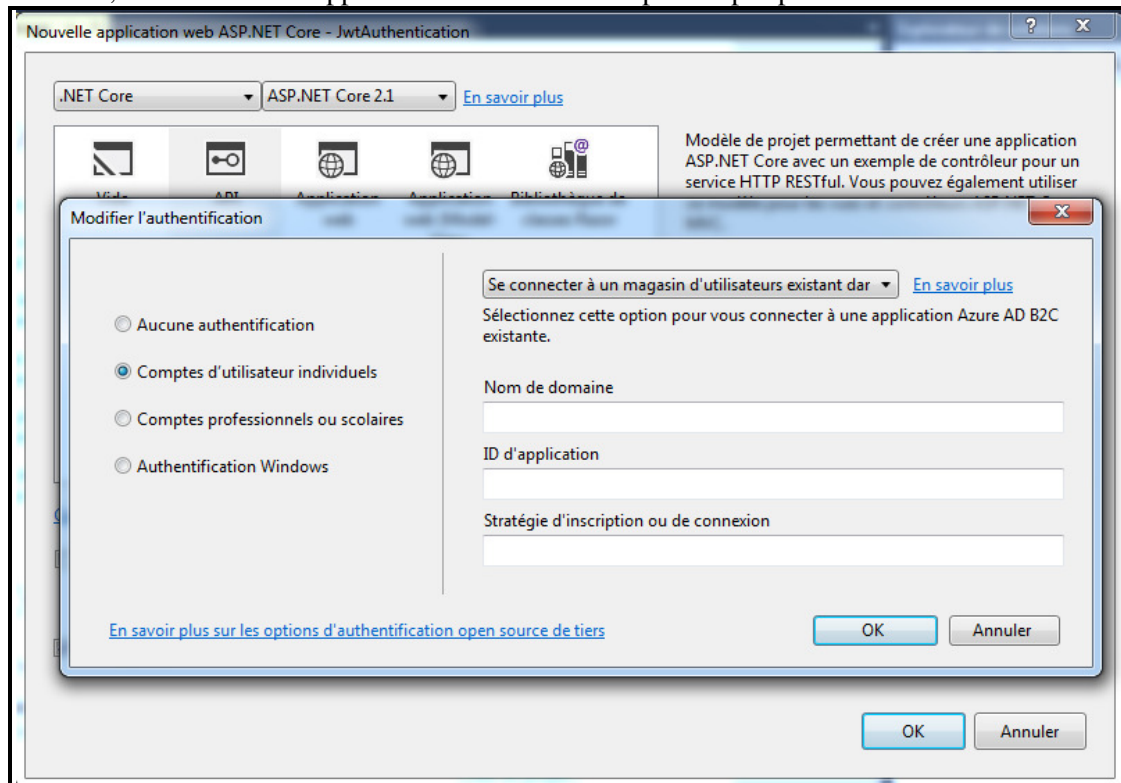
Rappelons que le template qu'on a opté pour c'était les API.

Rappelons que dans l'explorateur de la solution, nous disposons d'une arborescence incluant deux dossiers dont le premier correspond aux contrôleurs et le second est relatif aux modèles ainsi que le contexte de la base de données.



2

Si par exemple, on avait créé une application web et qu'on a choisi le template API puis on a sélectionné l'option d'authentification, on ne va nous demander de spécifier le nom de domaine, l'identifiant de l'application dont nous ne disposons pas pour l'instant.



On va donc créer nous-mêmes tout ce qu'il faut pour assurer l'authentification du backend de l'application RestaurantApp.

Nous allons opter pour une authentification via des autorisations à partir de la base de données. Nous disposons, si vous vous en souvenez, d'une classe appelée RestaurantDbContext.cs

```
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

namespace WebAPI_MasterDetail.Models
{
    public partial class RestaurantDbContext : DbContext
    {
        public RestaurantDbContext()
        {
        }

        public RestaurantDbContext(DbContextOptions<RestaurantDbContext>
options)
        : base(options)
        {
        }

        public virtual DbSet<Customer> Customer { get; set; }
    }
}
```

```
public virtual DbSet<Item> Item { get; set; }
public virtual DbSet<Order> Order { get; set; }
public virtual DbSet<OrderItems> OrderItems { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        #warning To protect potentially sensitive information in your connection
string, you should move it out of source code. See
http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing
connection strings.

optionsBuilder.UseSqlServer("Server=(localdb)\\v11.0;Database=RestaurantDB;Trus
ted_Connection=True;");

//optionsBuilder.UseSqlServer("Server=tcp:mpssidbserver2.database.windows.net,1
433;Initial Catalog=mpssi_db;Persist Security Info=False;User
ID=HamdiBEJI@mpssidbserver2;Password=hamdi110901@yahoo.fr;MultipleActiveResultS
ets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;");
    }
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Customer>(entity =>
    {
        entity.Property(e => e.CustomerId)
            .HasColumnName("CustomerId")
            .ValueGeneratedNever();

        entity.Property(e => e.Name)
            .HasMaxLength(50)
            .IsUnicode(false);
    });

    modelBuilder.Entity<Item>(entity =>
    {
        entity.Property(e => e.ItemId).HasColumnName("ItemId");

        entity.Property(e => e.Name)
            .HasMaxLength(50)
            .IsUnicode(false);
    });

    modelBuilder.Entity<Order>(entity =>
    {
        entity.Property(e => e.OrderId).HasColumnName("OrderId");

        entity.Property(e => e.CustomerId).HasColumnName("CustomerId");

        entity.Property(e => e.Gtotal).HasColumnName("GTotal");
    });
}
```

4

```
entity.Property(e => e.OrderNo).HasMaxLength(50);

entity.Property(e => e.Pmethod)
    .HasColumnName("PMethod")
    .HasMaxLength(50)
    .IsUnicode(false);

entity.HasOne(d => d.Customer)
    .WithMany(p => p.Order)
    .HasForeignKey(d => d.CustomerId)
    .HasConstraintName("FK_Order_Customer");
});

modelBuilder.Entity<OrderItems>(entity =>
{
    entity.HasKey(e => e.OrderItemId);

    entity.Property(e =>
e.OrderItemId).HasColumnName("OrderItemId");

    entity.Property(e => e.ItemId).HasColumnName("ItemId");

    entity.Property(e => e.OrderId).HasColumnName("OrderId");

    entity.HasOne(d => d.Item)
        .WithMany(p => p.OrderItems)
        .HasForeignKey(d => d.ItemId)
        .HasConstraintName("FK_OrderItems_Item");

    entity.HasOne(d => d.Order)
        .WithMany(p => p.OrderItems)
        .HasForeignKey(d => d.OrderId)
        .HasConstraintName("FK_OrderItems_Order");
});
}
}
```

De petites modifications devraient être attribuées à cette classe de manière à ce qu'elle puisse supporter l'authentification.

Pour ce faire, nous lui avons attribué ces changements

```
using System;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

namespace WebAPI_MasterDetail.Models
{
    public partial class RestaurantDbContext : IdentityDbContext<IdentityUser>
    {
        public RestaurantDbContext()
        {

```

```
}

public RestaurantDbContext(DbContextOptions<RestaurantDbContext>
options)
    : base(options)
{
}
public virtual DbSet<Customer> Customer { get; set; }
public virtual DbSet<Item> Item { get; set; }
public virtual DbSet<Order> Order { get; set; }
public virtual DbSet<OrderItems> OrderItems { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        #warning To protect potentially sensitive information in your connection
string, you should move it out of source code. See
http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing
connection strings.

optionsBuilder.UseSqlServer("Server=(localdb)\\v11.0;Database=RestaurantDB;Trus
ted_Connection=True;");

//optionsBuilder.UseSqlServer("Server=tcp:mpssidbserver2.database.windows.net,1
433;Initial Catalog=mpssi_db;Persist Security Info=False;User
ID=HamdiBEJI@mpssidbserver2;Password=hamdi110901@yahoo.fr;MultipleActiveResultS
ets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;");
    }
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    #region Seed Data
    modelBuilder.Entity<IdentityRole>().HasData(
        new { Id = 1, Name = "Admin", NormalizedName = "ADMIN" },
        new { Id = 2, Name = "Customer", NormalizedName = "CUSTOMER" }
    );
    #endregion

    modelBuilder.Entity<Customer>(entity =>
    {
        entity.Property(e => e.CustomerId)
            .HasColumnName("CustomerId")
            .ValueGeneratedNever();

        entity.Property(e => e.Name)
            .HasMaxLength(50)
            .IsUnicode(false);
    });
    modelBuilder.Entity<Item>(entity =>
```

6

```
{
    entity.Property(e => e.ItemId).HasColumnName("ItemId");

    entity.Property(e => e.Name)
        .HasMaxLength(50)
        .IsUnicode(false);
});
modelBuilder.Entity<Order>(entity =>
{
    entity.Property(e => e.OrderId).HasColumnName("OrderId");

    entity.Property(e => e.CustomerId).HasColumnName("CustomerId");

    entity.Property(e => e.Gtotal).HasColumnName("GTotal");

    entity.Property(e => e.OrderNo).HasMaxLength(50);

    entity.Property(e => e.Pmethod)
        .HasColumnName("PMethod")
        .HasMaxLength(50)
        .IsUnicode(false);

    entity.HasOne(d => d.Customer)
        .WithMany(p => p.Order)
        .HasForeignKey(d => d.CustomerId)
        .HasConstraintName("FK_Order_Customer");
});

modelBuilder.Entity<OrderItems>(entity =>
{
    entity.HasKey(e => e.OrderItemId);

    entity.Property(e =>
e.OrderItemId).HasColumnName("OrderItemId");

    entity.Property(e => e.ItemId).HasColumnName("ItemId");

    entity.Property(e => e.OrderId).HasColumnName("OrderId");

    entity.HasOne(d => d.Item)
        .WithMany(p => p.OrderItems)
        .HasForeignKey(d => d.ItemId)
        .HasConstraintName("FK_OrderItems_Item");

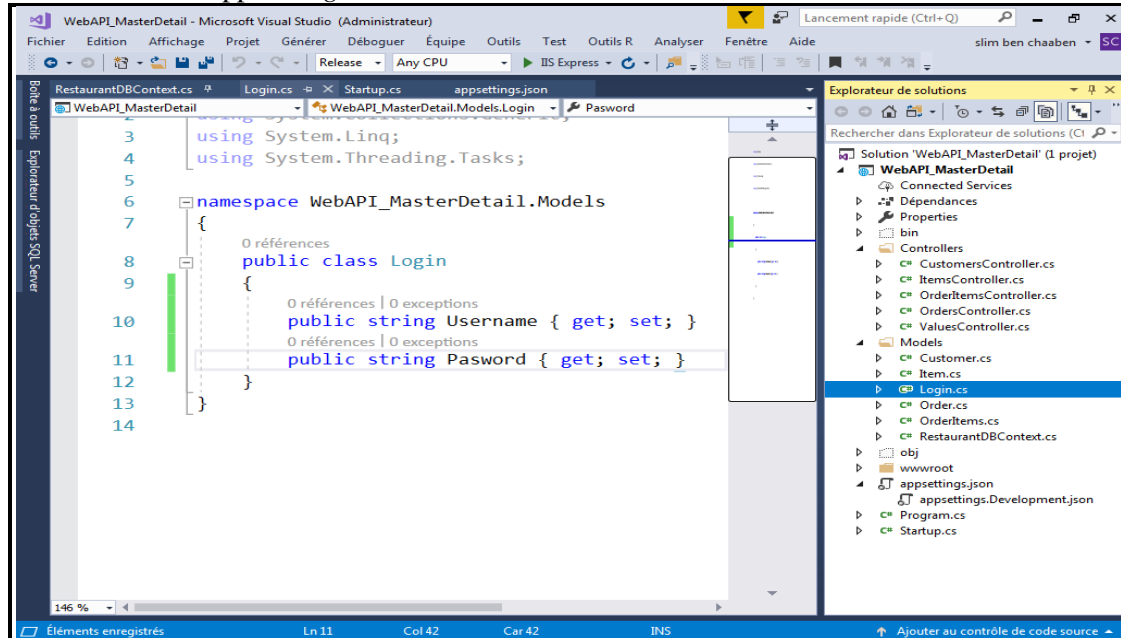
    entity.HasOne(d => d.Order)
        .WithMany(p => p.OrderItems)
        .HasForeignKey(d => d.OrderId)
        .HasConstraintName("FK_OrderItems_Order");
});
}
}
```

7

Puisque les API sont non visuels, si nous souhaitons nous authentifier à travers ce service de Web API, nous devrions passer un username et un password.

C'est la raison pour laquelle, nous devrions créer le modèle approprié pour *LOGIN*

Cette classe sera appelé *Login* tout court



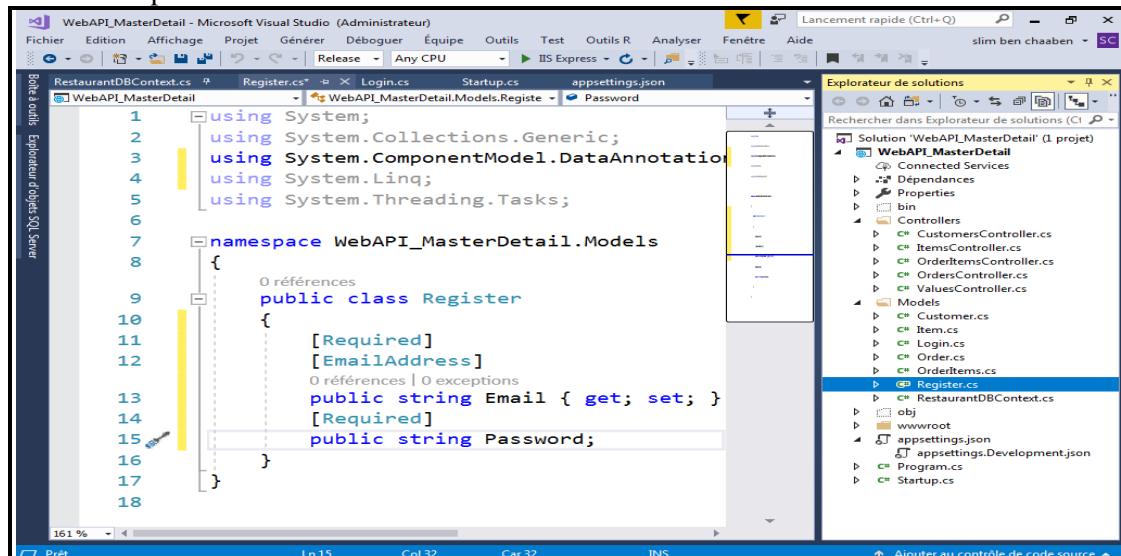
```
1 using System.Linq;
2 using System.Threading.Tasks;
3
4 namespace WebAPI_MasterDetail.Models
5 {
6     0 références
7     public class Login
8     {
9         0 références | 0 exceptions
10        public string Username { get; set; }
11        0 références | 0 exceptions
12        public string Password { get; set; }
13    }
14 }
```

Les propriétés Username et Password sont importantes parce que quand vous vous authentifiez à travers Postman ou à travers un serveur client tels que Angular, React, Android, etc, vous serez amenés à introduire votre nom d'utilisateur et votre mot de passe.

Le deuxième point qu'il faut assurer serait évidemment l'inscription (REGISTER) donc si vous n'avez pas de username et de password, nous vous autorisons de vous inscrire.

Pour cela, on va ajouter un nouveau modèle que l'on nommera Register

Le code C# pour ce ViewModel est le suivant :



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel.DataAnnotations;
4 using System.Linq;
5 using System.Threading.Tasks;
6
7 namespace WebAPI_MasterDetail.Models
8 {
9     0 références
10    public class Register
11    {
12        [Required]
13        [EmailAddress]
14        0 références | 0 exceptions
15        public string Email { get; set; }
16        [Required]
17        public string Password;
18    }
19 }
```

Hamdi BEJI

[hamdi.beji@isg.rnu.tn](mailto:hamdi.beji@isg.rnu.tn)

[hamdi.beji@sesame.com.tn](mailto:hamdi.beji@sesame.com.tn)

8

Quand vous vous inscrivez, on vous demande de faire l'enregistrement avec votre email qui sera par la suite votre nom d'utilisateur.

La propriété Email est annotée avec l'annotation [EmailAddress] donc le modèle se chargera de la valider tout en sachant qu'elle est obligatoire [Required] et pareillement pour la propriété Password.

Nous aurons aussi besoin d'une chaîne de connexion dont nous nous disposons déjà.

On aura aussi la possibilité de l'ajouter dans le fichier appsettings.json comme suit:

```
{
  "ConnectionStrings": {
    "DefaultConnection":
      "Server=(localdb)\\v11.0;Database=RestaurantDB;Trusted_Connection=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Comme on l'a déjà vu auparavant, (localdb)\\v11.0 est le nom de serveur, RestaurantDB est le nom de la base de données.

On va introduire dans cette même branche une section pour JWT comme suit:

```
{
  "ConnectionStrings": {
    "DefaultConnection":
      "Server=(localdb)\\v11.0;Database=RestaurantDB;Trusted_Connection=True;"
  },
  "JWT": {
    "Site": "http://www.isg.rnu.tn/",
    "SigningKey": "Tunis Paris Berlin Cairo Sydney Tokyo Beijing Roma London Athens",
    "ExpiryInMinutes": "60"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Il s'agit d'un groupe d'éléments de configuration de JWT.

Site pourrait le site URL de l'organisation à laquelle vous appartenez.

Dans SigningKey, j'ai mis les noms de quelques villes

Le token pourrait expirer dans 60 minutes (ExpiryInMinutes)



9

Si vous désirez que ça expire après des jours, vous pouvez convertir ce nombre en heures puis les multiplier par 60 pour avoir la durée totale en minutes.

Nous devrions ensuite à la même occasion associer la classe de contexte de la base de données avec la chaîne de connexion et ça se fait d'ailleurs dans le fichier Startup.cs dans la méthode ConfigureServices()

Ci-après le code qui permet d'associer le *DbContext* avec la *ConnectionString*

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<RestaurantDbContext>(
        option =>
        option.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1).
        AddJsonOptions(
            options => {
                var resolver = options.SerializerSettings.ContractResolver;
                if(resolver!=null)
                    (resolver as
                    DefaultContractResolver).NamingStrategy=null;
            });
    services.AddCors();
}
```

Nous spécifions ici que nous utilisons *SqlServer* et nous récupérons le chaîne de connexion *DefaultConnection* à partir du fichier *appsettings.json*

Nous recourrons après à *IdentityFramework* pour faire l'authentification.

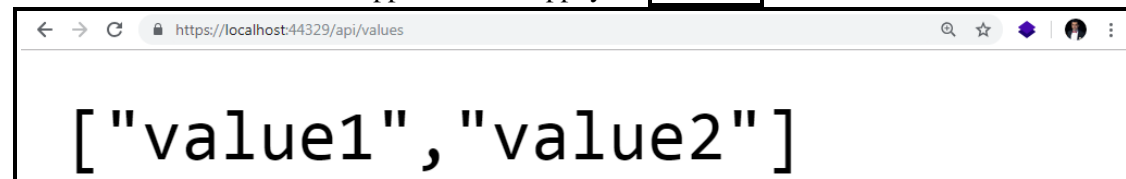
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<RestaurantDbContext>(
        option =>
        option.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<IdentityUser, IdentityRole>(
        option =>
        {
            option.Password.RequireDigit = false;
            option.Password.RequiredLength = 6;
            option.Password.RequireNonAlphanumeric = false;
            option.Password.RequireUppercase = false;
            option.Password.RequireLowercase = false;
        }
    ).AddEntityFrameworkStores<RestaurantDbContext>()
    .AddDefaultTokenProviders();
}
```

10

```
services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1).
    AddJsonOptions(
        options=> {
            var resolver = options.SerializerSettings.ContractResolver;
            if(resolver!=null)
                (resolver as
DefaultContractResolver).NamingStrategy=null;
        });
services.AddCors();
}
```

Allons maintenant tester notre application en appuyant **Ctrl + F5**

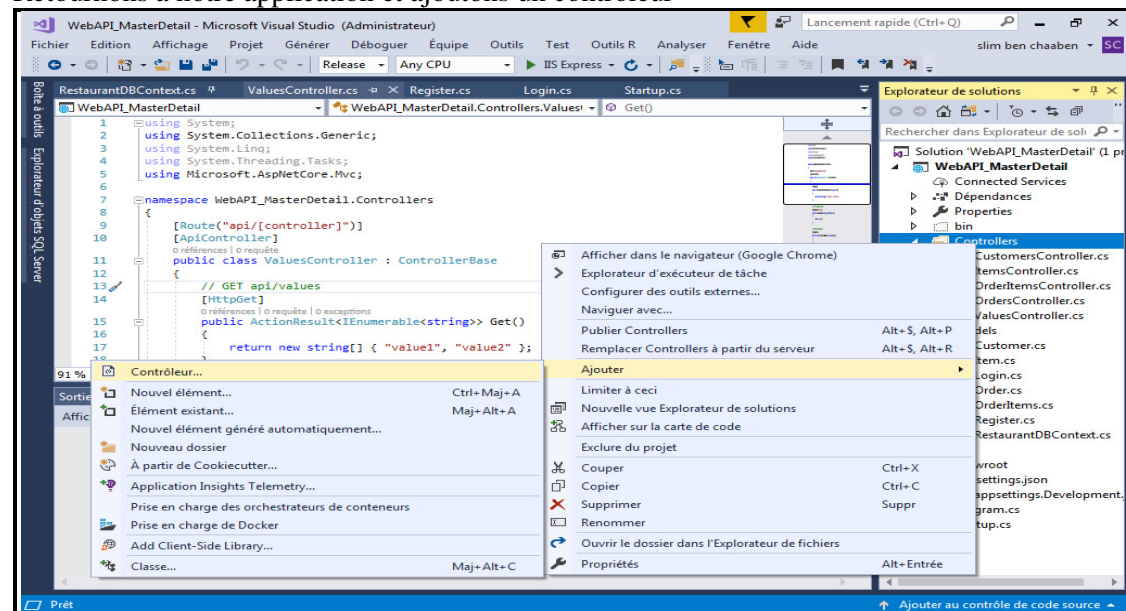


Nous avons ainsi des valeurs émises à partir de ValuesController.

Ce que nous désirons faire c'est que vous ne verrez pas ces valeurs si vous ne vous authentifiez pas. C'est d'ailleurs l'objectif de ce tutoriel.

Pour ce faire, nous avons besoin d'un contrôleur d'authentification. Ce contrôleur d'authentification sera responsable pour enregistrer un nouvel utilisateur et pour authentifier cet utilisateur et notamment générer un Token.

Retournons à notre application et ajoutons un contrôleur



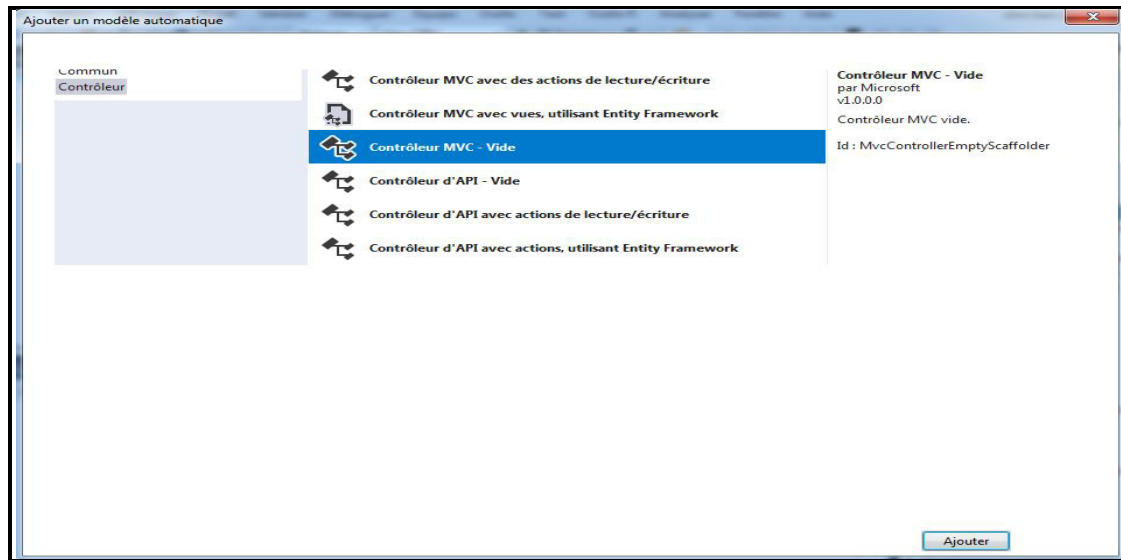
Ce contrôleur sera vide

Hamdi BEJI

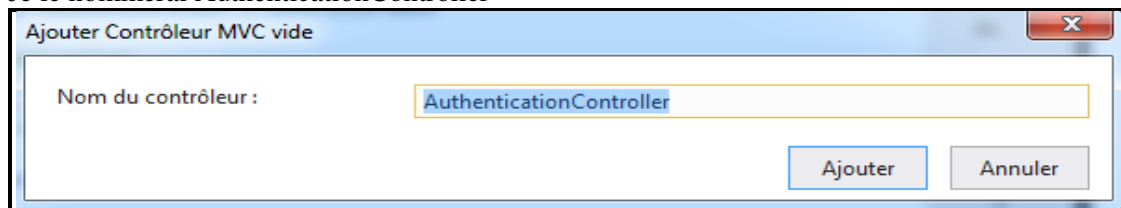
[hamdi.beji@isg.rnu.tn](mailto:hamdi.beji@isg.rnu.tn)

[hamdi.beji@sesame.com.tn](mailto:hamdi.beji@sesame.com.tn)

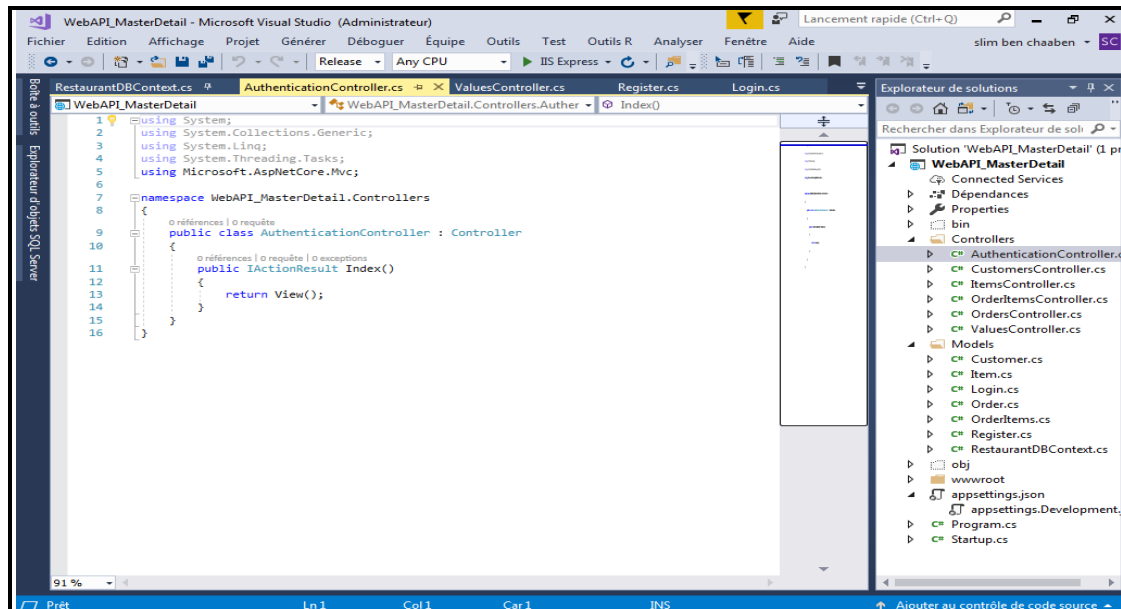
11



Je le nommerai AuthenticationController



On obtient du code dans ce contrôleur



Nous avons essentiellement deux principales méthodes d'actions dont la première correspond à l'enregistrement et la seconde correspond à l'authentification proprement dite.

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using WebAPI_MasterDetail.Models;

namespace WebAPI_MasterDetail.Controllers
{
    public class AuthenticationController : Controller
    {
        private readonly UserManager<IdentityUser> _userManager;
        private readonly IConfiguration _configuration;

        public AuthenticationController(UserManager<IdentityUser> userManager,
        IConfiguration configuration)
        {
            _userManager = userManager;
            _configuration = configuration;
        }
    }
}
```

```
[Route("register")]
[HttpPost]
public async Task<ActionResult> InsertUser([FromBody] Register model)
{
    var user = new IdentityUser
    {
        Email = model.Email,
        UserName = model.Email,
        SecurityStamp = Guid.NewGuid().ToString()
    };
    var result = await _userManager.CreateAsync(user, model.Password);
    if(result.Succeeded)
    {
        await _userManager.AddToRoleAsync(user, "Customer");
    }
    return Ok(new { Username = user.UserName });
}
```

```
[Route("login")]
[HttpPost]
public async Task<ActionResult> Login([FromBody] Login model)
{
    var user = await _userManager.FindByNameAsync(model.Username);
}
```

13

```
        if(user!=null && await
_userManager.CheckPasswordAsync(user,model.Pasword))
        {
            var claim = new[]
            {
                new Claim(JwtRegisteredClaimNames.Sub,user.UserName)
            };
            var signingKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(_configuration["Jwt:SigningKey"]));

            int expiryInMinutes=
Convert.ToInt32(_configuration["Jwt:ExpiryInMinutes"]);

            var token = new JwtSecurityToken(
                issuer: _configuration["Jwt:Site"],
                audience: _configuration["Jwt:Site"],
                expires: DateTime.UtcNow.AddMinutes(expiryInMinutes),
                signingCredentials: new SigningCredentials(signingKey,
SecurityAlgorithms.HmacSha256));

            return Ok(
                new
                {
                    token = new
JwtSecurityTokenHandler().WriteToken(token),
                    expiration = token.ValidTo
                });
        }
        return Unauthorized();
    }
}
```

Dans ce code, nous avons deux points d'arrivée.

Le premier c'est d'ajouter un user que nous pouvons tout simplement spécifier à l'aide du endpoint **register**.

Cela veut dire que pour arriver à ce point, il suffit de taper **/register** ans pour autant taper **api/register**

Le second endpoint est également **/login**

Cela nous permet de passer le nom d'utilisateur et le mot de passe et ce pour l'authentification.

Il existe des paires d'éléments dont nous aurons besoin pour générer notre Token

Le Token exige un **SigningKey** qui est secret.

Il exige le temps d'expiration en minutes (**ExpiryInMinutes**)

Nous avons également les deux propriétés

```
        issuer: _configuration["Jwt:Site"],
        audience: _configuration["Jwt:Site"],
```

14

Notons que ces données proviennent d'une branche dans le fichier appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection":
    "Server=(localdb)\\v11.0;Database=RestaurantDB;Trusted_Connection=True;"
  },
  "Jwt": {
    "Site": "http://www.isg.rnu.tn/",
    "SigningKey": "Tunis Paris Berlin Cairo Sydney Tokyo Beijing Roma London Athens",
    "ExpiryInMinutes": "60"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Interpétons maintenant les deux méthodes.

Quand nous enregistrons un user qui est identifié par un email et un password.

```
[Required]
[EmailAddress]
public string Email { get; set; }
[Required]
public string Password;
```

Nousinstancions un objet de *IdentityUser* avec l'Email, le Password provenant du modèle passé déjà en paramètre dans la méthode InsertUser()

Nous créons ensuite un user en utilisant *\_userManager* qu'on a injecté dans le constructeur du contrôleur ainsi que *\_configuration*

```
public AuthenticationController(UserManager<IdentityUser> userManager,
IConfiguration configuration)
```

C'est ce qu'on appelle une injection de dépendances.

*\_configuration* nous aide à récupérer les clés provenant du fichier appsettings.json

```
var result = await _userManager.CreateAsync(user, model.Password);
```

Ce code permet de créer un nouvel utilisateur étant donné l'utilisateur ainsi que son mot de passe que l'utilisateur avait choisi.

15

```
if(result.Succeeded)
{
    await _userManager.AddToRoleAsync(user, "Customer");
}
```

Si le résultat d'ajout est réussi, on ajoute cet utilisateur au rôle de **Customer**  
Rappelez-vous qu'on a ajouté ce rôle dans la classe `RestaurantDbContext` au début de ce tutoriel.

On a essentiellement deux rôles, celui de l'administrateur et celui du client.  
A chaque fois qu'on a un utilisateur à enregistrer, on va lui attribuer le rôle du client.

```
return Ok(new { Username = user.UserName });
```

On retourne enfin le nouveau nom d'utilisateur.

Nous devrions ajouter du code à la méthode `ConfigureServices(IServiceCollection services)` pour dire que nous utilisons *Token Authentication*

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<RestaurantDbContext>(
        option =>
        option.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<IdentityUser, IdentityRole>(
        option =>
        {
            option.Password.RequireDigit = false;
            option.Password.RequiredLength = 6;
            option.Password.RequireNonAlphanumeric = false;
            option.Password.RequireUppercase = false;
            option.Password.RequireLowercase = false;
        }
    ).AddEntityFrameworkStores<RestaurantDbContext>()
    .AddDefaultTokenProviders();
```

```
services.AddAuthentication(
    option =>
    {
        option.DefaultAuthenticateScheme =
        JwtBearerDefaults.AuthenticationScheme;
        option.DefaultChallengeScheme =
        JwtBearerDefaults.AuthenticationScheme;
        option.DefaultScheme =
        JwtBearerDefaults.AuthenticationScheme;
    }).AddJwtBearer(options =>
    {
        options.SaveToken = true;
        options.RequireHttpsMetadata = true;
        options.TokenValidationParameters = new
        TokenValidationParameters()
```

```
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidAudience = Configuration["Jwt:Site"],
            ValidIssuer = Configuration["Jwt:Site"],
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:SigningKey"]))
        };
    };
};

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1).
    AddJsonOptions(
        options=> {
            var resolver = options.SerializerSettings.ContractResolver;
            if(resolver!=null)
                (resolver as
DefaultContractResolver).NamingStrategy=null;
        });
    services.AddCors();
}
```

L'objectif c'était d'ajouter une authentification à cette application.  
Le schéma d'authentification par défaut est JWT

Dans la méthode Configure(IApplicationBuilder app, IHostingEnvironment env)  
on va indiquer que l'on va utiliser le service d'authentification

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.Use(async (context, next) =>
    {
        await next();
        if(context.Response.StatusCode==404 &&
!Path.HasExtension(context.Request.Path.Value))
        {
            context.Request.Path = "/index.html";
            await next();
        }
    });
    app.UseDefaultFiles();
    app.UseStaticFiles();
    /*else
    {
        app.UseHsts();
    }*/
    app.UseAuthentication();
    //app.UseHttpsRedirection();
    app.UseCors(options =>
```

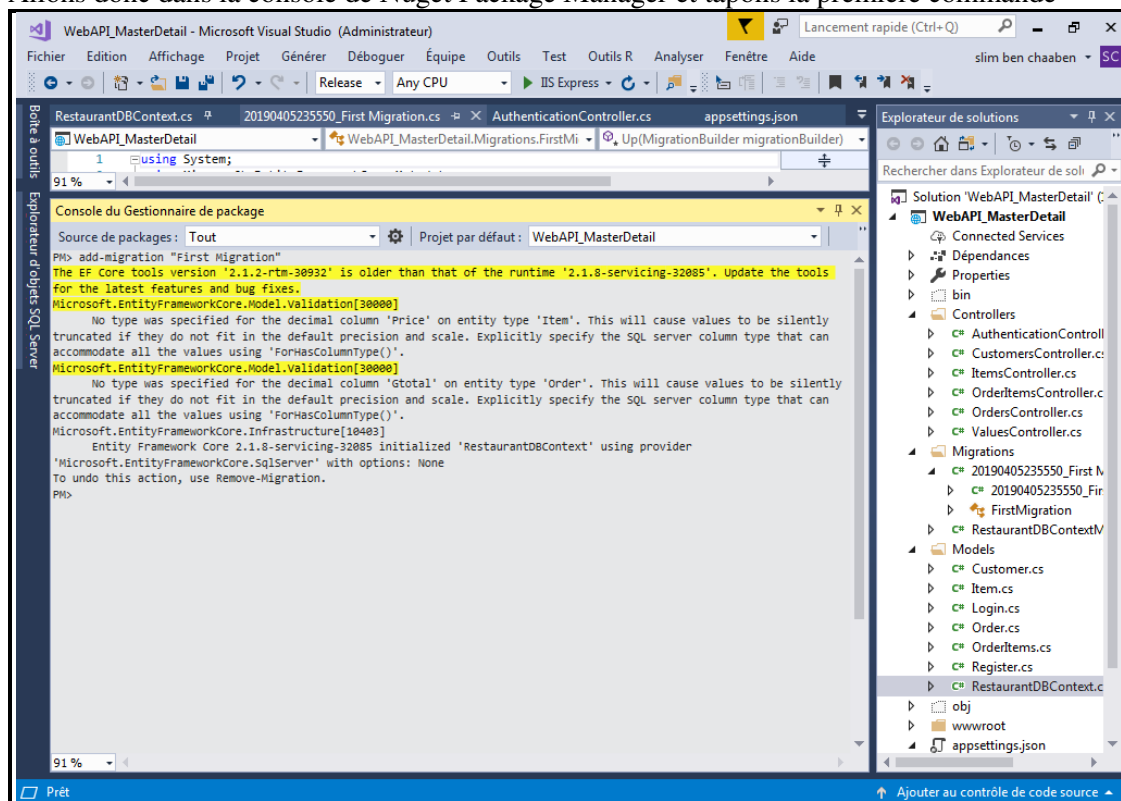


17

```
//options.WithOrigins("https://mpssi.azurewebsites.net").  
options.WithOrigins("https://localhost:4200").  
//options.WithOrigins("https://localhost:44329").  
AllowAnyMethod().  
AllowAnyHeader();  
app.UseMvc();  
}
```

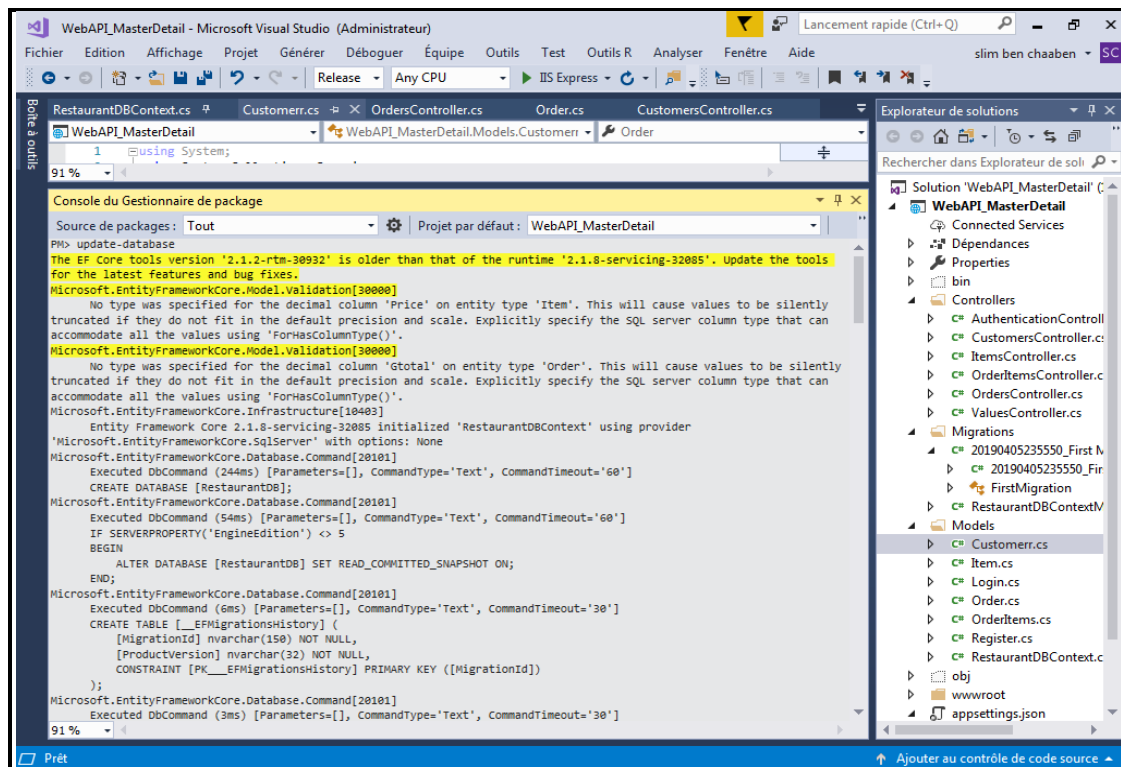
Nous devrions ensuite faire une migration vers la base de données compte tenu de ces configurations qu'on a faites

Allons donc dans la console de Nuget Package Manager et tapons la première commande



Tapons ensuite la commande update-database

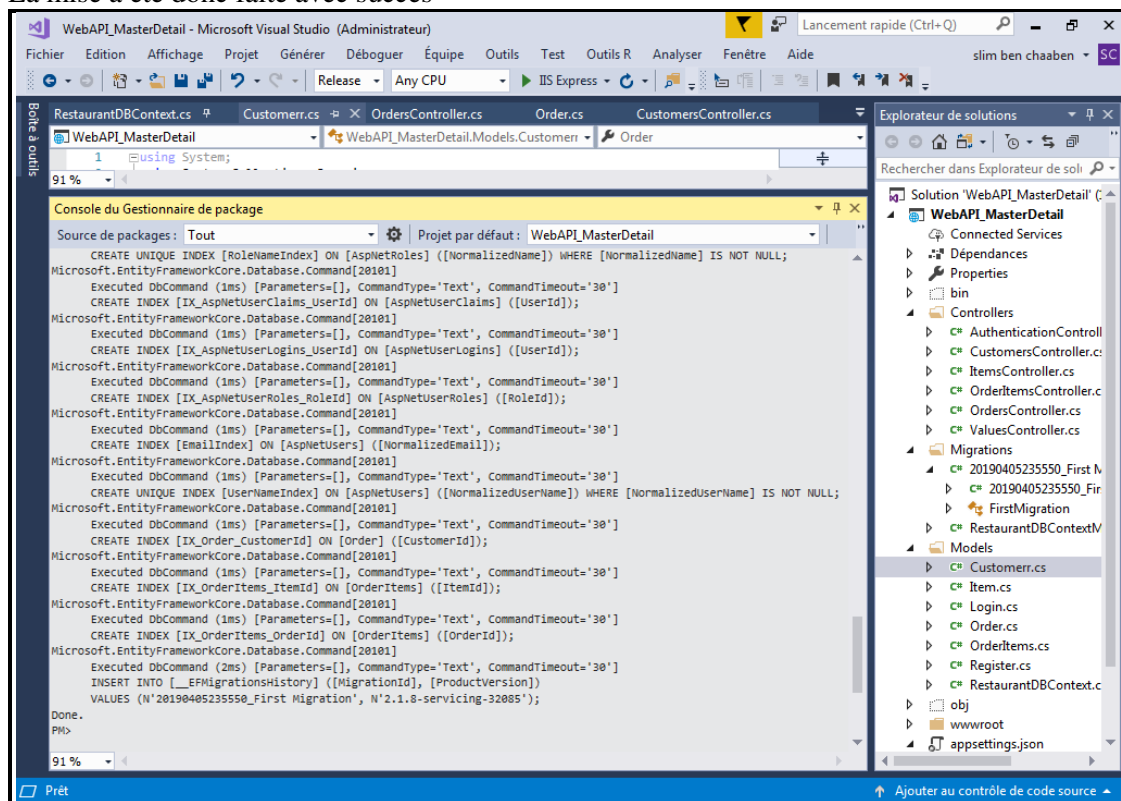
18



The screenshot shows the Package Manager Console in Visual Studio. The output displays the execution of database commands for creating a database, tables, and indexes. The Explorer pane on the right shows the project structure.

```
PM> update-database
The EF Core tools version '2.1.2-rtm-30932' is older than that of the runtime '2.1.8-servicing-32085'. Update the tools for the latest features and bug fixes.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
No type was specified for the decimal column 'Price' on entity type 'Item'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
No type was specified for the decimal column 'Gtotal' on entity type 'Order'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
Entity Framework Core 2.1.8-servicing-32085 initialized 'RestaurantDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (244ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
CREATE DATABASE [RestaurantDB];
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (54ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
IF SERVERPROPERTY('EngineEdition') <> 5
BEGIN
ALTER DATABASE [RestaurantDB] SET READ_COMMITTED_SNAPSHOT ON;
END;
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (6ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [__EFMigrationsHistory] (
[MigrationId] nvarchar(150) NOT NULL,
[ProductVersion] nvarchar(32) NOT NULL,
CONSTRAINT [PK__EFMigrationsHistory] PRIMARY KEY ([MigrationId])
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
```

La mise a été donc faite avec succès

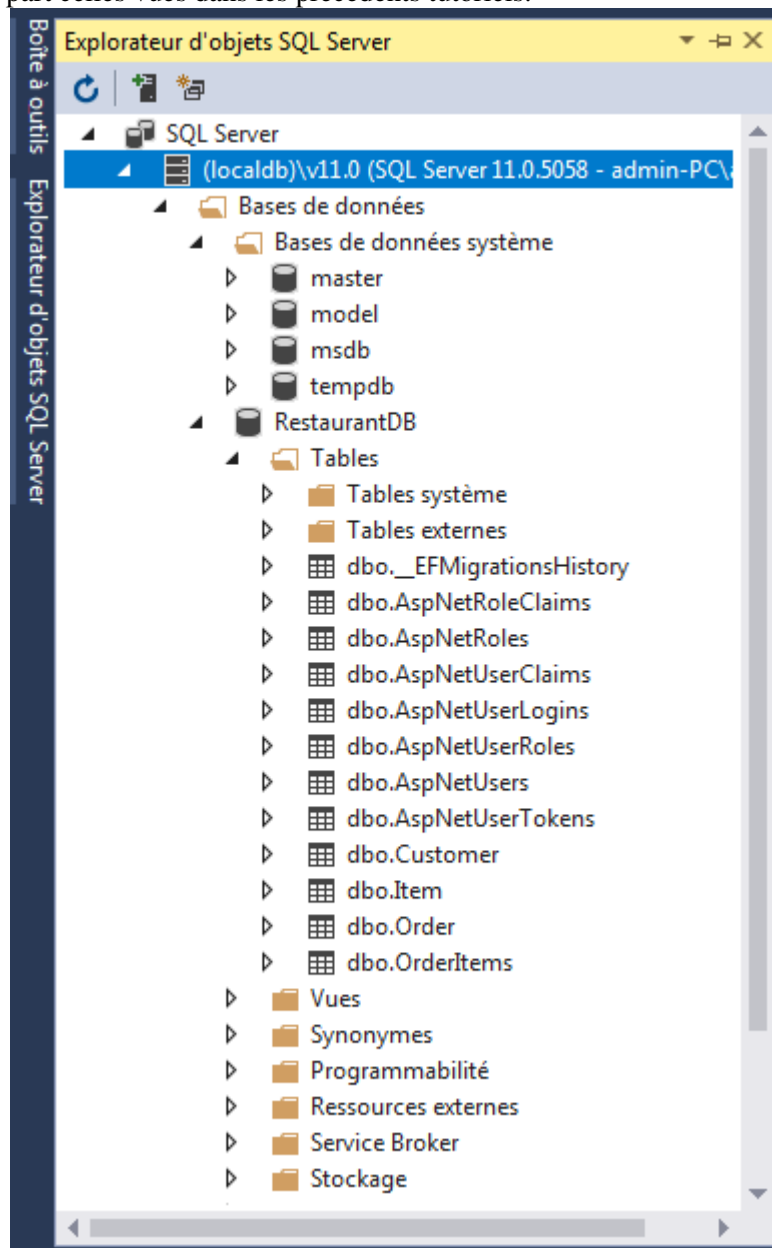


The screenshot shows the Package Manager Console in Visual Studio. The output displays the execution of database commands for creating a database, tables, and indexes. The Explorer pane on the right shows the project structure.

```
CREATE UNIQUE INDEX [RoleNameIndex] ON [AspNetRoles] ([NormalizedUserName]) WHERE [NormalizedUserName] IS NOT NULL;
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_AspNetUserClaims_UserId] ON [AspNetUserClaims] ([UserId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_AspNetUserLogins_UserId] ON [AspNetUserLogins] ([UserId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_AspNetUserRoles_RoleId] ON [AspNetUserRoles] ([RoleId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [EmailIndex] ON [AspNetUsers] ([NormalizedEmail]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE UNIQUE INDEX [UserNameIndex] ON [AspNetUsers] ([NormalizedUserName]) WHERE [NormalizedUserName] IS NOT NULL;
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_Order_CustomerId] ON [Order] ([CustomerId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_OrderItems_ItemId] ON [OrderItems] ([ItemId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_OrderItems_OrderId] ON [OrderItems] ([OrderId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20190405235550_First Migration', N'2.1.8-servicing-32085');
```

Hamdi BEJI  
[hamdi.beji@isg.rnu.tn](mailto:hamdi.beji@isg.rnu.tn)  
[hamdi.beji@sesame.com.tn](mailto:hamdi.beji@sesame.com.tn)

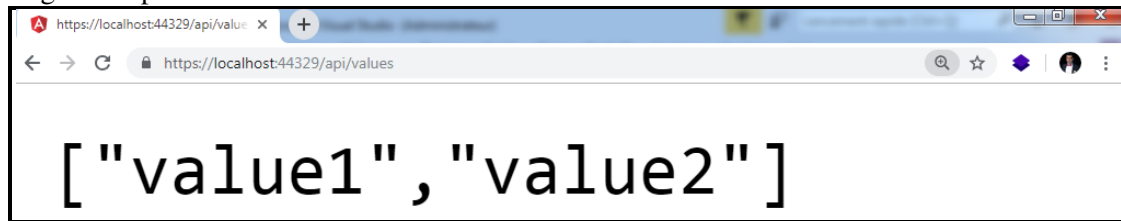
A ce niveau, nous devrions avoir une base de données contenant des tables supplémentaire à part celles vues dans les précédents tutoriels.



Nous avons obtenu en plus toutes les tables de EntityFramework

20

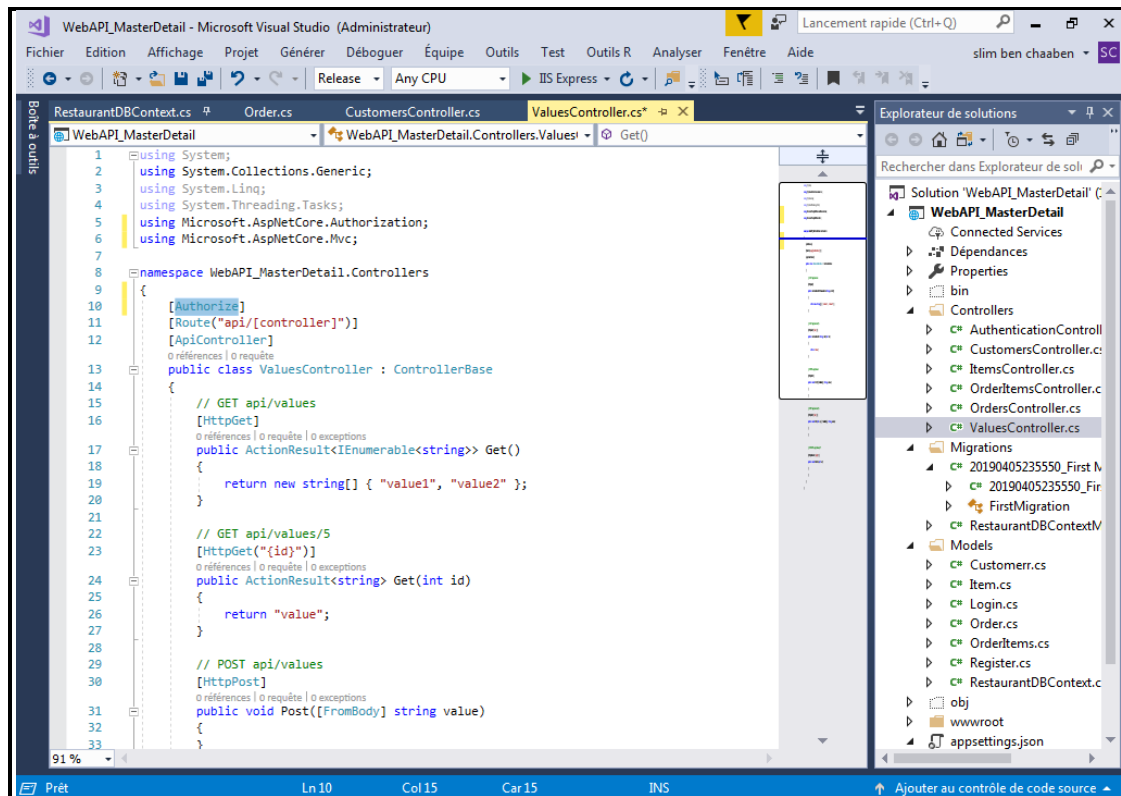
Regénrons puis exécutons



Nous obtenons désespérément les mêmes valeurs que toute à l'heure.

Pour appliquer les restrictions de l'authentification JWT, il suffit d'ajouter l'annotation [Authorize] au-dessus de chaque contrôleur.

Allons dans le contrôleur ValuesController



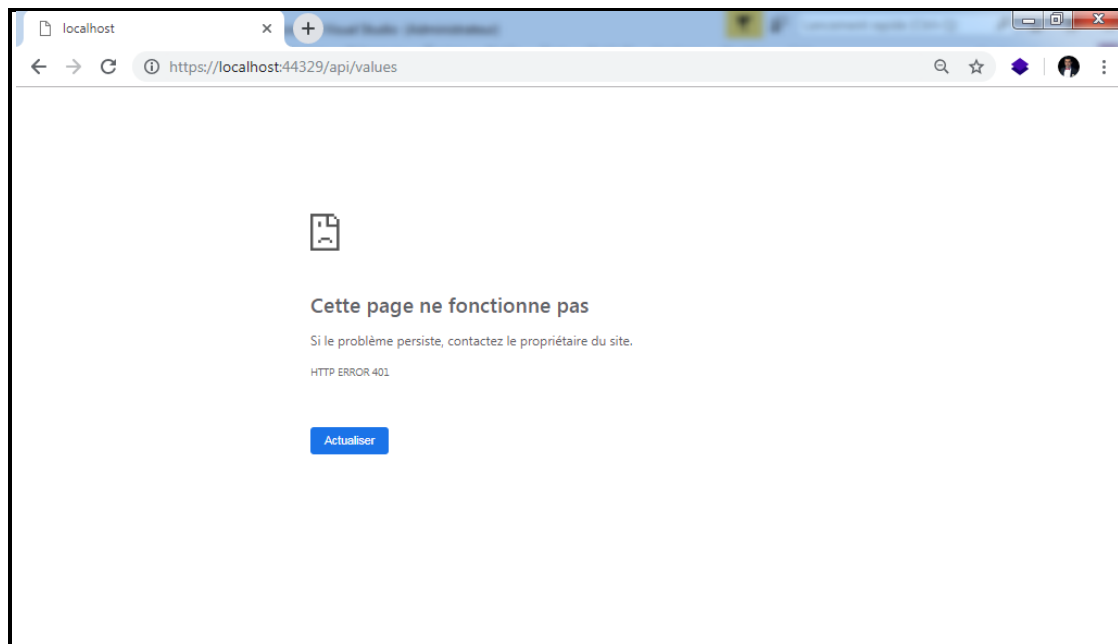
Appuyons Ctrl+F5 et revoyons le résultat maintenant

Hamdi BEJI

[hamdi.beji@isg.rnu.tn](mailto:hamdi.beji@isg.rnu.tn)

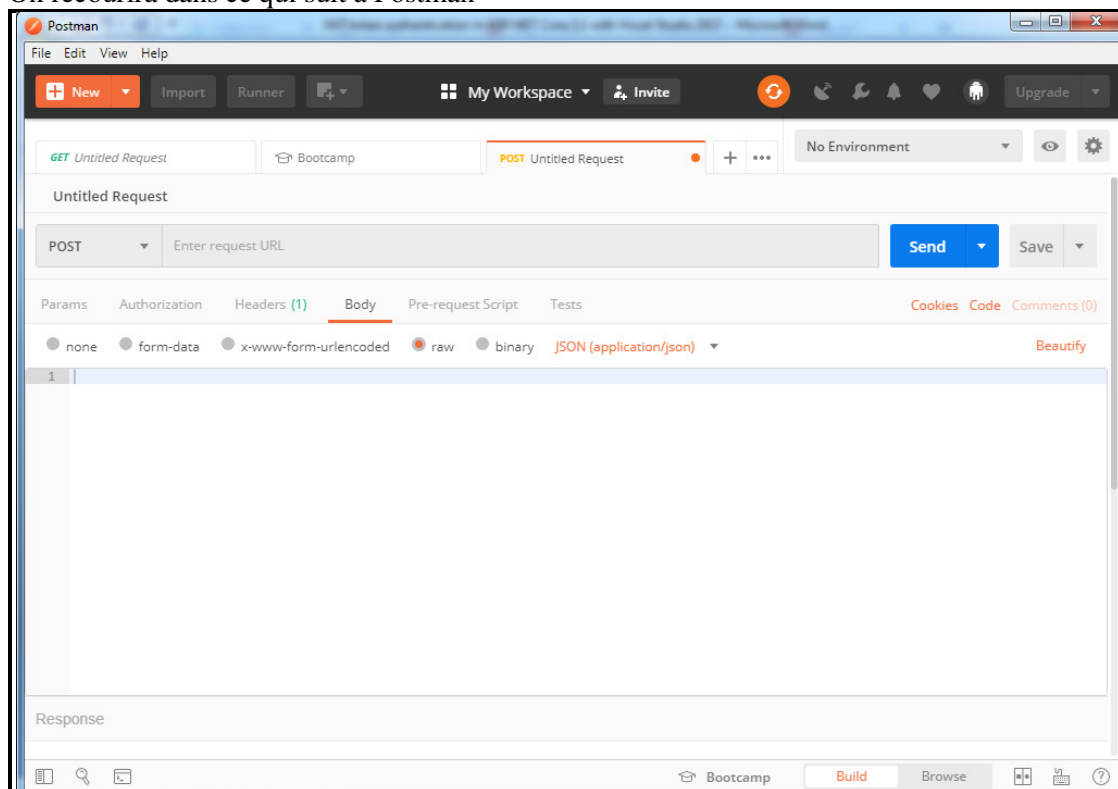
[hamdi.beji@sesame.com.tn](mailto:hamdi.beji@sesame.com.tn)

21



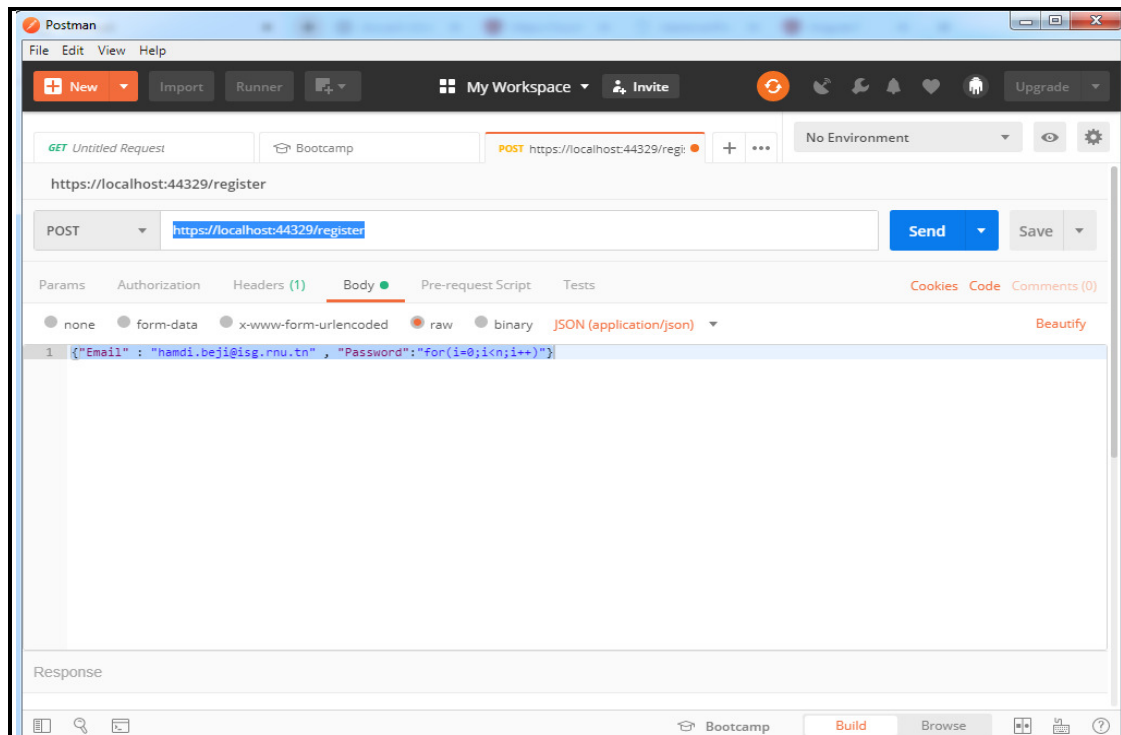
On obtient cette fois-ci l'erreur HTTP ERROR 401 qui veut dire que vous n'êtes pas autorisé.  
Pour remédier à ce souci, nous devrions nous enregistrer tout d'abord.

On recourra dans ce qui suit à Postman



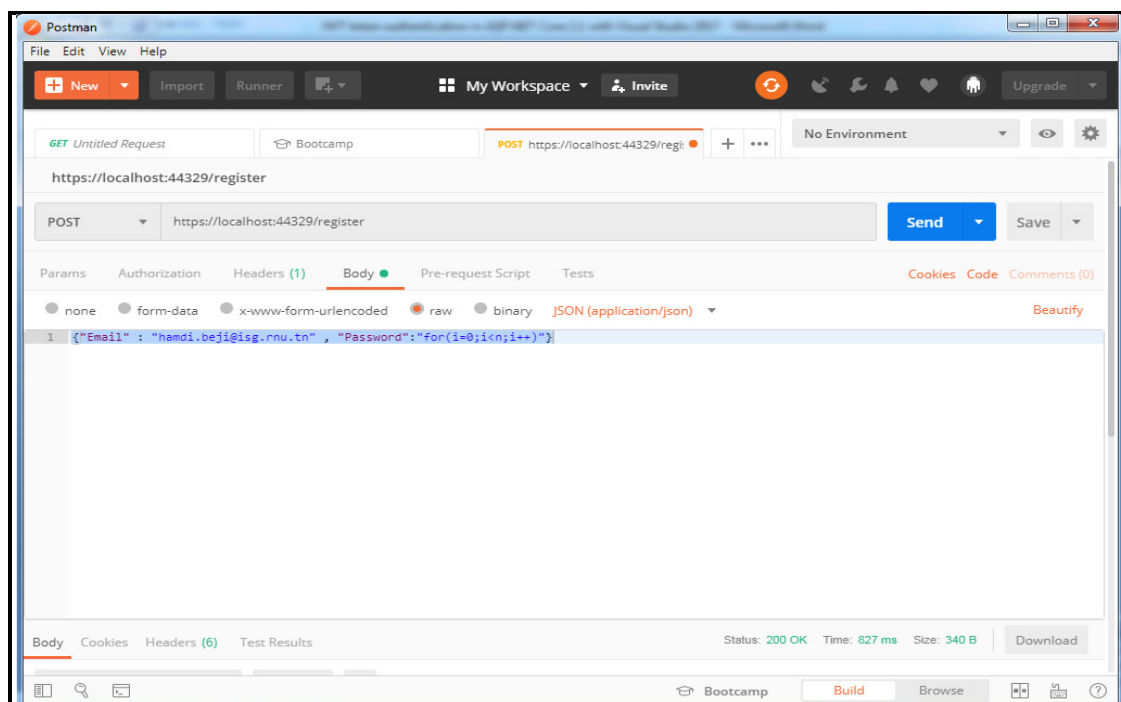
22

Je choisis POST comme request et dans le Body je vais envoyer un raw d'un objet de type JSON



Je vais maintenant envoyer ma requête au serveur.

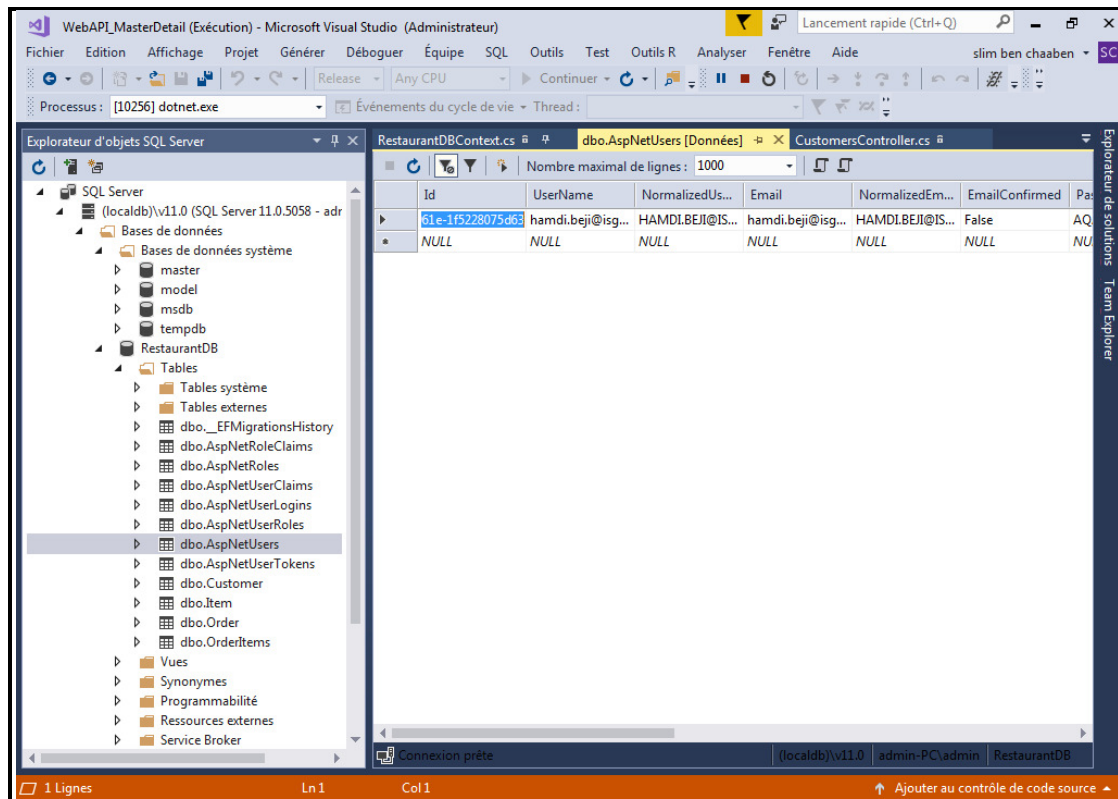
On obtient une réponse de statut 200 OK



Hamdi BEJI  
[hamdi.beji@isg.rnu.tn](mailto:hamdi.beji@isg.rnu.tn)  
[hamdi.beji@sesame.com.tn](mailto:hamdi.beji@sesame.com.tn)

23

Cela veut dire que le user a été enregistré avec succès.  
 Revenons à la base de données et vérifions cela.



Allons maintenant nous authentifier mais avant de le faire, allons comprendre ce que fait la méthode `Login([FromBody] Login model)`  
 Le endpoint Login prend en paramètres un username et un password  
 On y cherche ensuite l'utilisateur par nom d'utilisateur  
 Si le résultat est non null et que le password coïncide bien avec celui de l'utilisateur passé en paramètre, on instancie un nouveau Claim se basant sur ce username.  
 On récupère ensuite la clé que l'on a dans le fichier de configuration.  
 On récupère aussi le temps d'expiration et on le convertit en entier.  
 On aura après le code qui sert à créer le Token de sécurité.

```
var token = new JwtSecurityToken(
    issuer: _configuration["Jwt:Site"],
    audience: _configuration["Jwt:Site"],
    expires: DateTime.UtcNow.AddMinutes(expiryInMinutes),
    signingCredentials: new SigningCredentials(signingKey,
    SecurityAlgorithms.HmacSha256))
```

On retourne après un objet de type JSON avec la propriété token et le temps d'expiration.

```
new
{
    token = new
    JwtSecurityTokenHandler().WriteToken(token),
    expiration = token.ValidTo
};
```

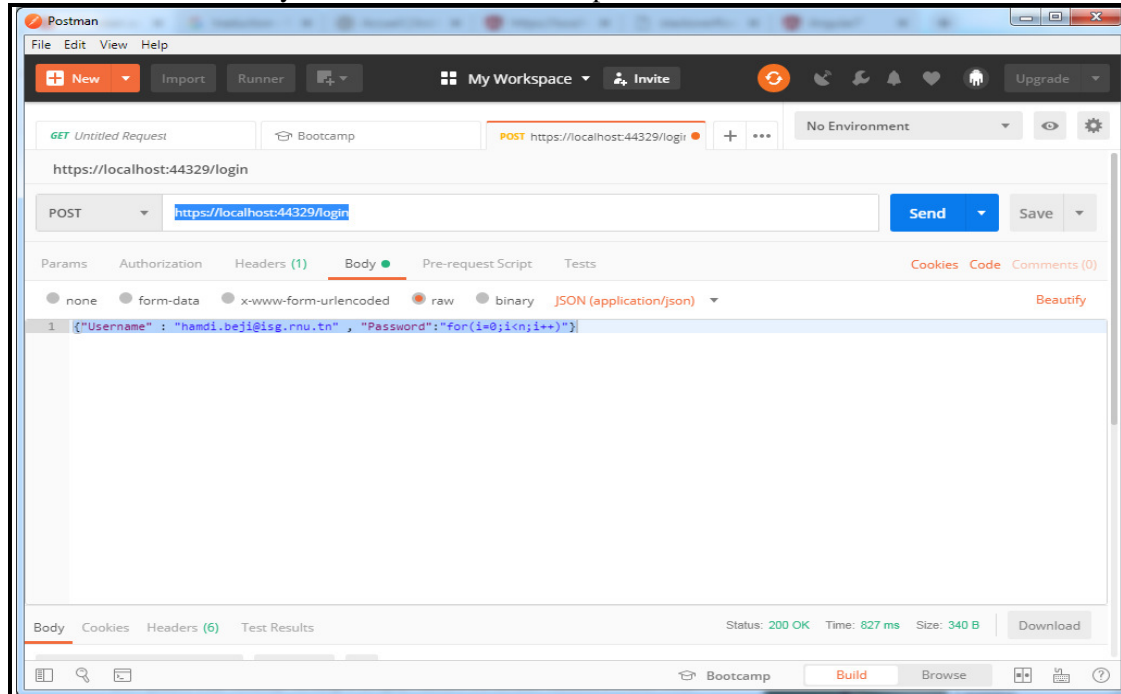


24

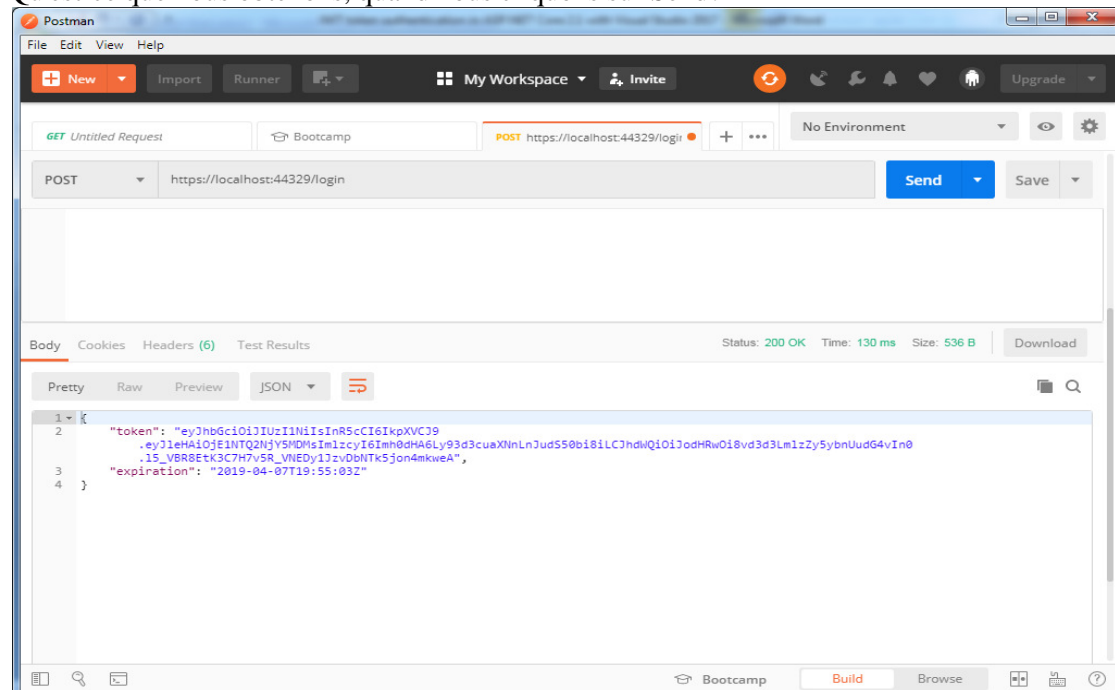
Si l'utilisateur n'est pas autorisé, on l'empêche.

```
return Unauthorized();
```

Allons maintenant essayer de nous authentifier depuis Postman



Qu'est-ce que nous obtenons, quand nous cliquons sur Send?

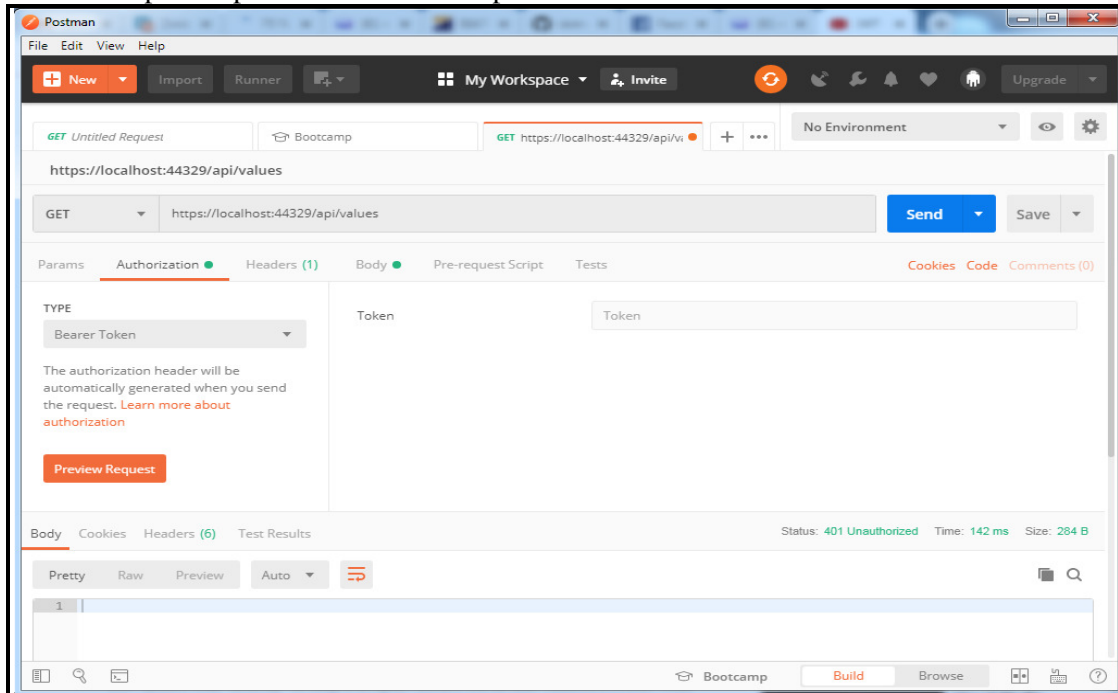


Nous obtenons notre token exactement comme on l'attendait.  
Nous avons ainsi un objet JSON avec deux propriétés.

Hamdi BEJI  
[hamdi.beji@isg.rnu.tn](mailto:hamdi.beji@isg.rnu.tn)  
[hamdi.beji@sesame.com.tn](mailto:hamdi.beji@sesame.com.tn)

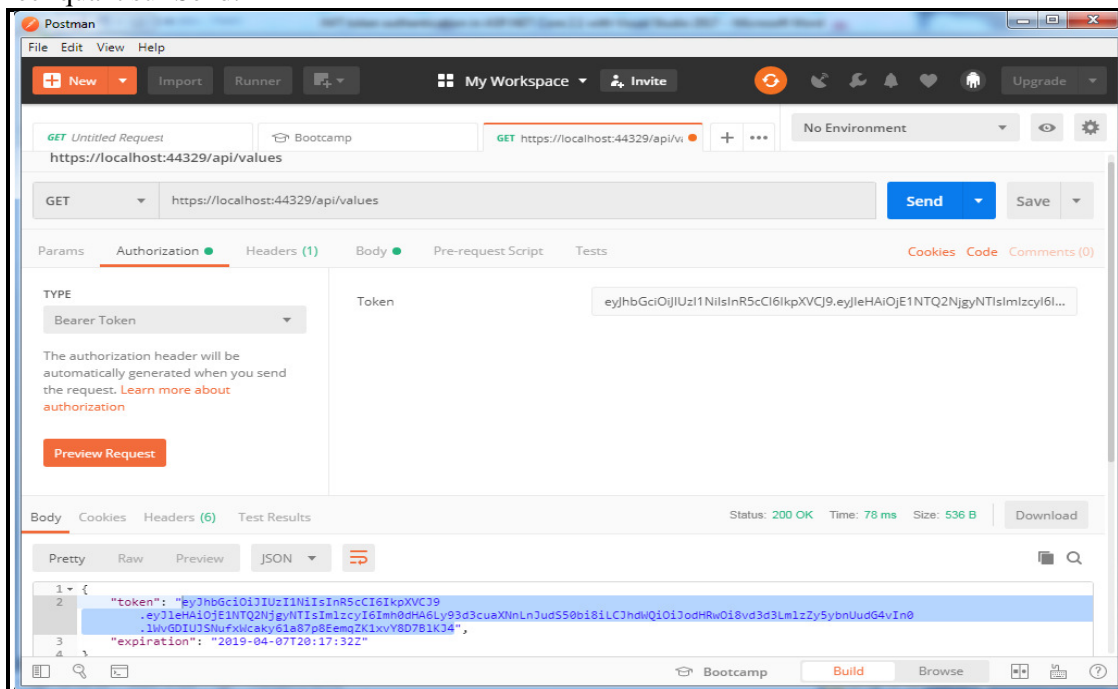


Maintenant, si l'on veut avoir accès aux données, nous devrions passer ce Token. Copions le Token dans la mémoire puis allons dans Postman pour récupérer par exemple values en tapant `https://localhost:44329/api/values`.



La réponse est Unauthorized (401)

Pour y remédier, ajoutons l'annotation [Authorized] dans ValuesController puis collons dans la zone Token de Bearer Token la valeur qu'on a copiée tout à l'heure puis réessayons en re cliquant sur Send.



---

Hamdi BEJI

hamdi.beji@isg.rnu.tn

hamdi.beji@sesame.com.tn

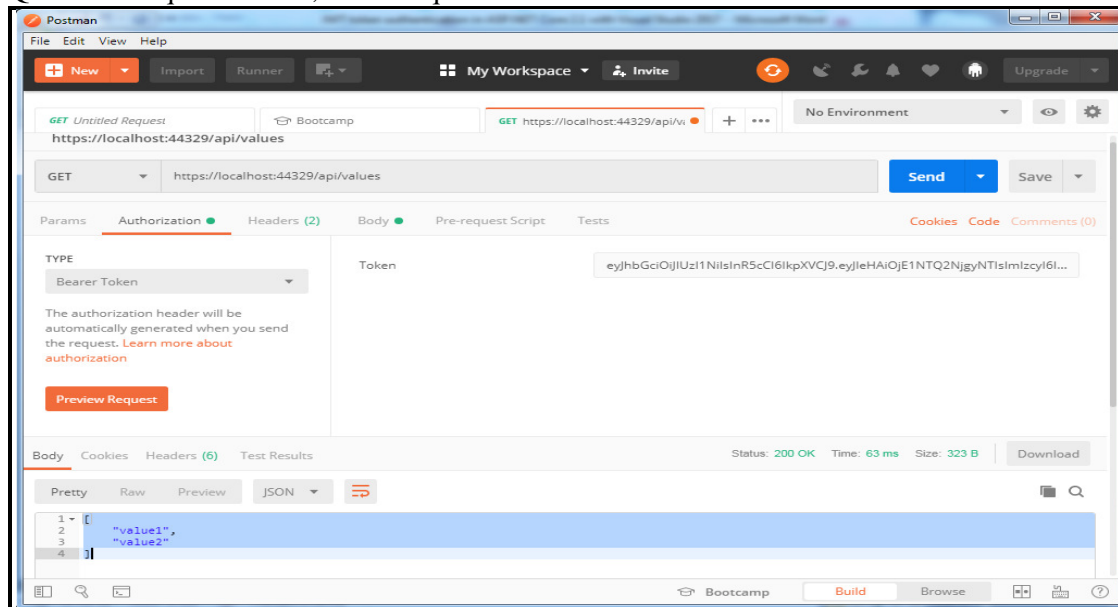
26

Rappelons qu'ici, on a choisi la méthode GET puis on tapé l'URI `https://localhost:44329/api/values`

Ensuite, on a sélectionné l'onglet Authorization et comme Type, on a opté pour Bearer Token.

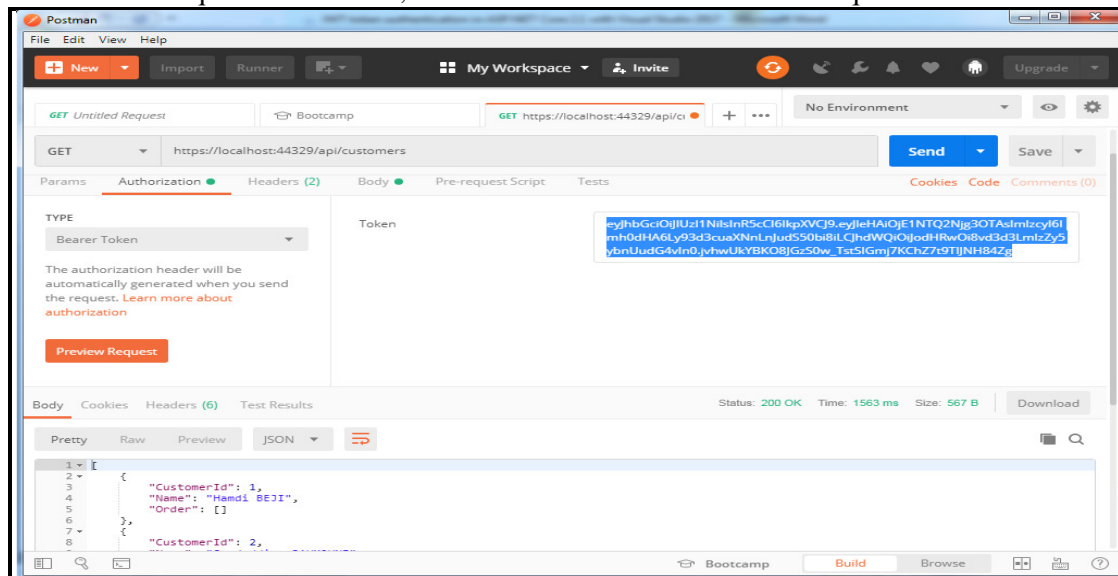
Dans la zone de saisie Token, on a collé le Token qu'on a copié tout à l'heure.

Quand on clique sur Send, voilà ce qu'on obtient comme résultat :



On obtient un résultat dont le statut est 200 OK avec les valeurs attendues ["value1", "value2"], ce qui prouve que la réponse était positive vu qu'on est devenus autorisés à accéder à ces données.

Pareillement, quand on tape l'URI `https://localhost:44329/api/customers` et ce après avoir mis l'annotation [Authorize] au-dessus de CustomersController ainsi que le Token avant d'effectuer la requête HTTP GET, on obtient la liste des Customers à laquelle on s'attendait



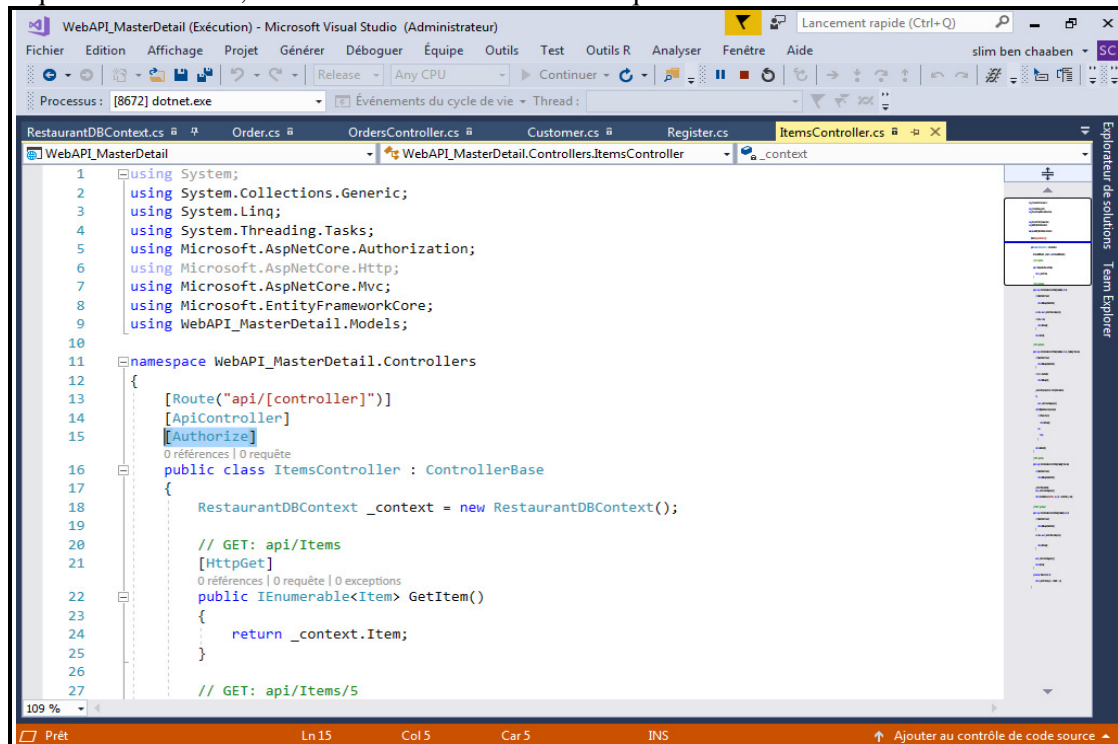
Hamdi BEJI

[hamdi.beji@isg.rnu.tn](mailto:hamdi.beji@isg.rnu.tn)

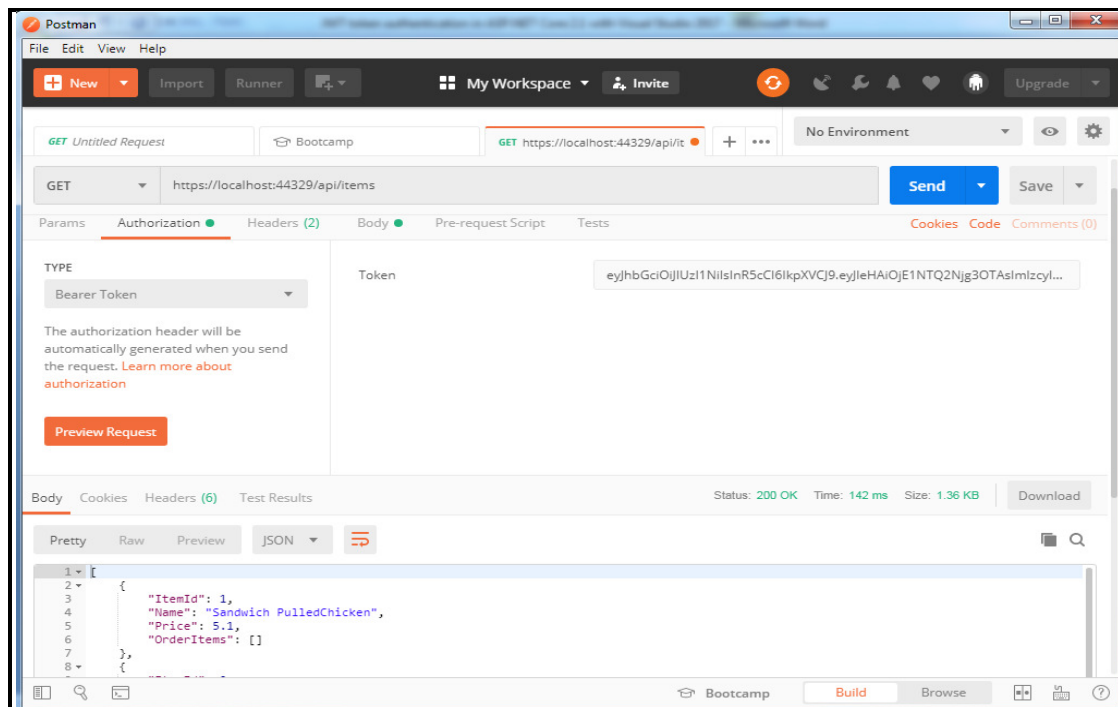
[hamdi.beji@sesame.com.tn](mailto:hamdi.beji@sesame.com.tn)

27

Pareillement, quand on tape l'URI `https://localhost:44329/api/items` et ce après avoir mis l'annotation `[Authorize]` au-dessus de `ItemsController` ainsi que le Token avant d'effectuer la requête HTTP GET, on obtient la liste des Items à laquelle on s'attendait



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Authorization;
6 using Microsoft.AspNetCore.Http;
7 using Microsoft.AspNetCore.Mvc;
8 using Microsoft.EntityFrameworkCore;
9 using WebAPI_MasterDetail.Models;
10
11 namespace WebAPI_MasterDetail.Controllers
12 {
13     [Route("api/[controller]")]
14     [ApiController]
15     [Authorize]
16     public class ItemsController : ControllerBase
17     {
18         RestaurantDBContext _context = new RestaurantDBContext();
19
20         // GET: api/Items
21         [HttpGet]
22         public IEnumerable<Item> GetItem()
23         {
24             return _context.Item;
25         }
26
27         // GET: api/Items/5
```



GET https://localhost:44329/api/items

Authorization: Bearer Token

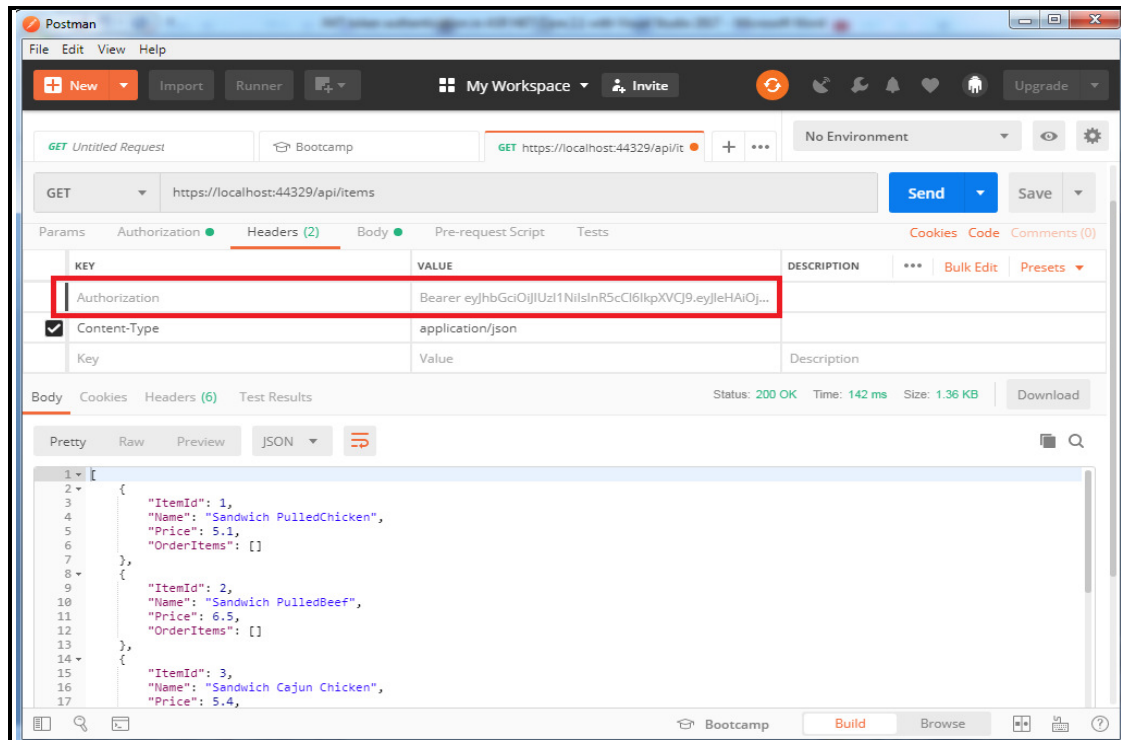
Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiJlNTQ2Njg3OTAsImVudCI6ImVudCJ9.eyJleHAiOiJlNTQ2Njg3OTAsImVudCI6ImVudCJ9

Status: 200 OK Time: 142 ms Size: 1.36 KB

```
1 [
2   {
3     "ItemId": 1,
4     "Name": "Sandwich PulledChicken",
5     "Price": 5.1,
6     "OrderItems": []
7   },
8   {
```

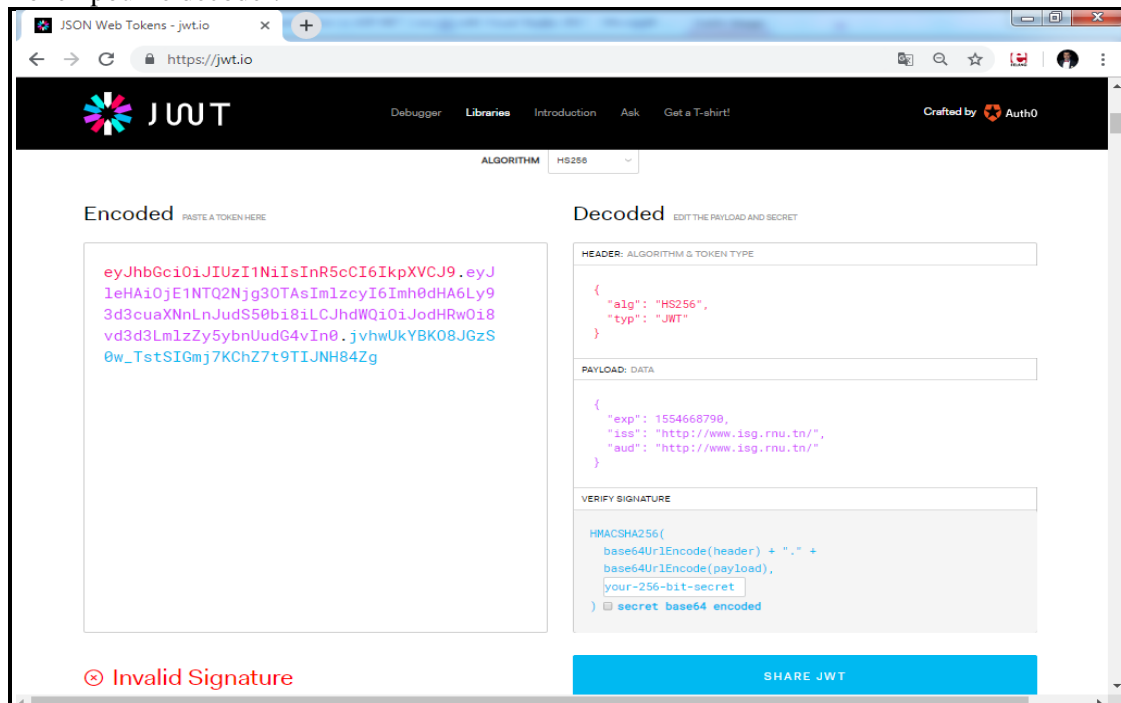
28

Allons maintenant dans Headers comme suit :



Nous voyons que la clé est Authorization et la valeur est le mot Bearer suivi d'un espace puis du Token.

Avec ce même Token, allons dans le site <https://jwt.io/> et faisons une rétro-ingénierie de ce Token pour le décoder.



29

Quand nous collons le Token dans l'espace réservé à gauche de l'écran, ça nous affiche à droite de l'écran quelques informations qui coïncident bien avec celles introduites dans notre backend.

A chaque couleur du Token encodé correspond une information de la même couleur dans la partie décodée.

Nous remarquons néanmoins que la signature est invalide.

⊗ Invalid Signature

Comment faire pour la rendre valide?

La réponse est tout simplement dans le fichier appsettings.json.

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\v11.0;Database=RestaurantDB;Trusted_Connection=True;"
  },
  "Jwt": {
    "Site": "http://www.isg.rnu.tn/",
    "SigningKey": "Tunis Paris Berlin Cairo Sydney Tokyo Beijing Roma London Athens",
    "ExpiryInMinutes": "60"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Il suffit de copier la liste des villes qu'on a mise dans SigningKey puis on la colle comme suit:

The screenshot shows the JWT.io website interface. On the left, under 'Encoded', there is a text area containing a JWT token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NTQ2Njg3OTAsImZcyI6Imh0dHA6Ly93d3cuXNnLnJudS00b08iLCJhdWQiOiJodHRwOi8vd3d3Lm1zZy5ybnUudG4vIn0.eyJvbmUkYBk08JGzS0w_TstSIGmj7KChZ7t9TIJNH84Zg`. On the right, under 'Decoded', the token is broken down into its parts: Header, Payload, and Signature. The payload is decoded to show its JSON structure: `{ "exp": 1554668790, "iss": "http://www.isg.rnu.tn/", "aud": "http://www.isg.rnu.tn/" }`. The signature is shown as `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), jJ Roma London Athens)`. At the bottom, it states 'Signature Verified' and has a 'SHARE JWT' button.

Dès que la clé de signature est collée dans l'endroit propice, la signature devient systématiquement valide

