Programmation avancée en c#.NET

Ing.Meryem OUARRACHI

Plan du module

Programmation WEB

- Généralités outils Web
- ASP MVC
- ASP.Net core
- Angular en ASP.Net Core

Génie logiciel en .Net

BI en Self Service

Programmation distribuée avancée

- Web API
- GraphQL

CHAPITRE 7:

.Net Core

Plan du chapitre

- .Net Core
- Docker
- ASP .Net Core
  - Développement sous linux
  - Entity Framework
  - Inversion du contrôle
  - Les sessions et les cookies
  - JWT

Gestion d’état

Cookies

-Création:

```
CookieOptions option = new CookieOptions();

option.HttpOnly = true;

 Response.Cookies.Append("Cookie1", value, option);
//ou bien Response.Cookies.Append("Cookie1", value);
```

-Récupération:

```
 String s =

this.ControllerContext.HttpContext.Request.Cookies["Cookie1"];
```

-Suppression

```
Response.Cookies.Delete("Cookie1");
```

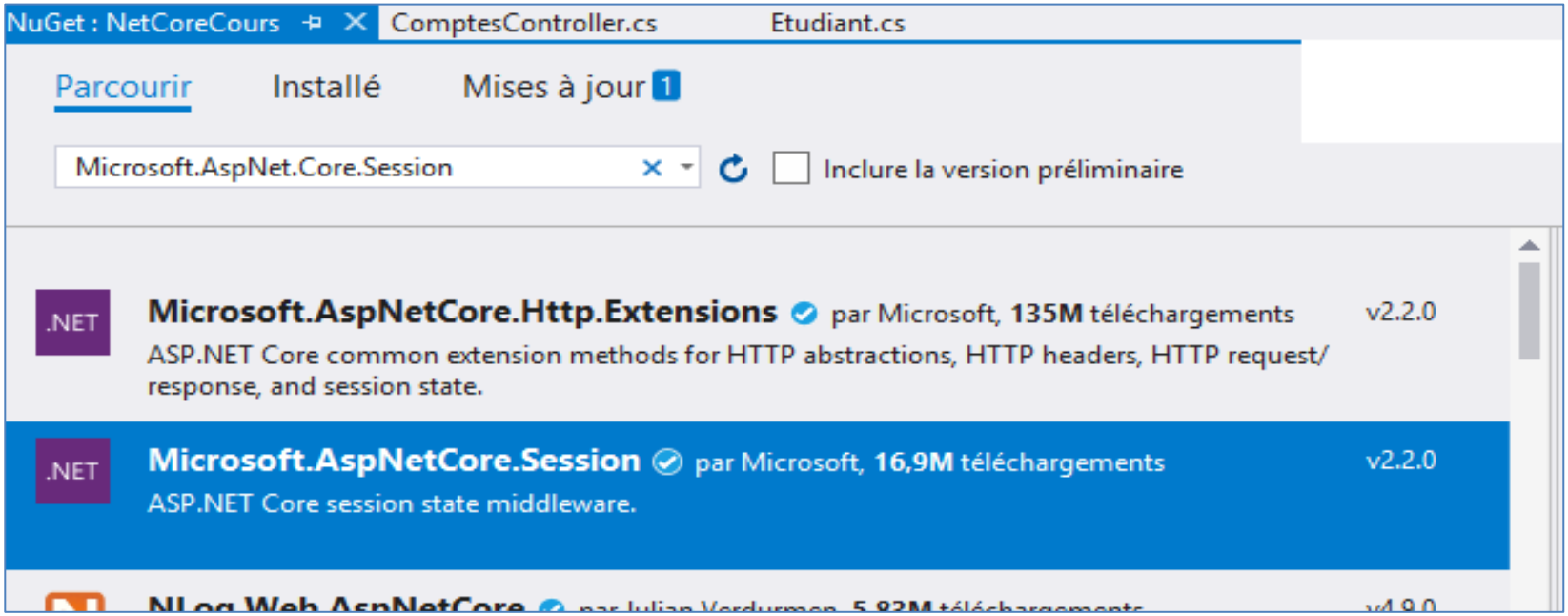
Cookies

-Vérification

this.ControllerContext.HttpContext.Request.Cookies["Cookie1"] != null

Sessions

1. Installer le package Microsoft.AspNetCore.Session



Sessions

2.Dans le fichier Startup: *ajouter le service de session*

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{   services.Configure<CookiePolicyOptions>(options =>
    {

        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddDistributedMemoryCache();

    services.AddSession();

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

Sessions

3.Dans le fichier Startup: *activer le service de session dans l'application*

```
// This method gets called by the runtime. Use this method to configure the HTTP
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();
    app.UseSession();

    app.UseMvc(routes =>
    {
```

Sessions

-Création:

```
HttpContext.Session.SetString("key1", value);

HttpContext.Session.SetInt32("key2",15);
```

-Récupération:

```
var name = HttpContext.Session.GetString ("key1");

var age = HttpContext.Session.GetInt32 ("key1");
```

Vérification

```
HttpContext.Session.GetString ("key1")!= null
```

Supprimer:

```
HttpContext.Session.Remove ("key1");
```

Inconvénients de sessions/cookies

1.Problème de sécurité

-Avoir des attaques CSRF et XSS si on n'oublie de prendre les précautions nécessaires

XSS	httpOnly=true et secure(https)
CSRF	ValidateAntiForgeryToken

Inconvénients de sessions/cookies

2.Problème de la montée en charge

Plus de charge sur le serveur →Problème de **la montée en charge**

-La montée en charge: Le concept de montée en charge consiste à qualifier l'architecture testée dans sa capacité à pouvoir supporter un nombre d'utilisateurs croissant sans défaillir.

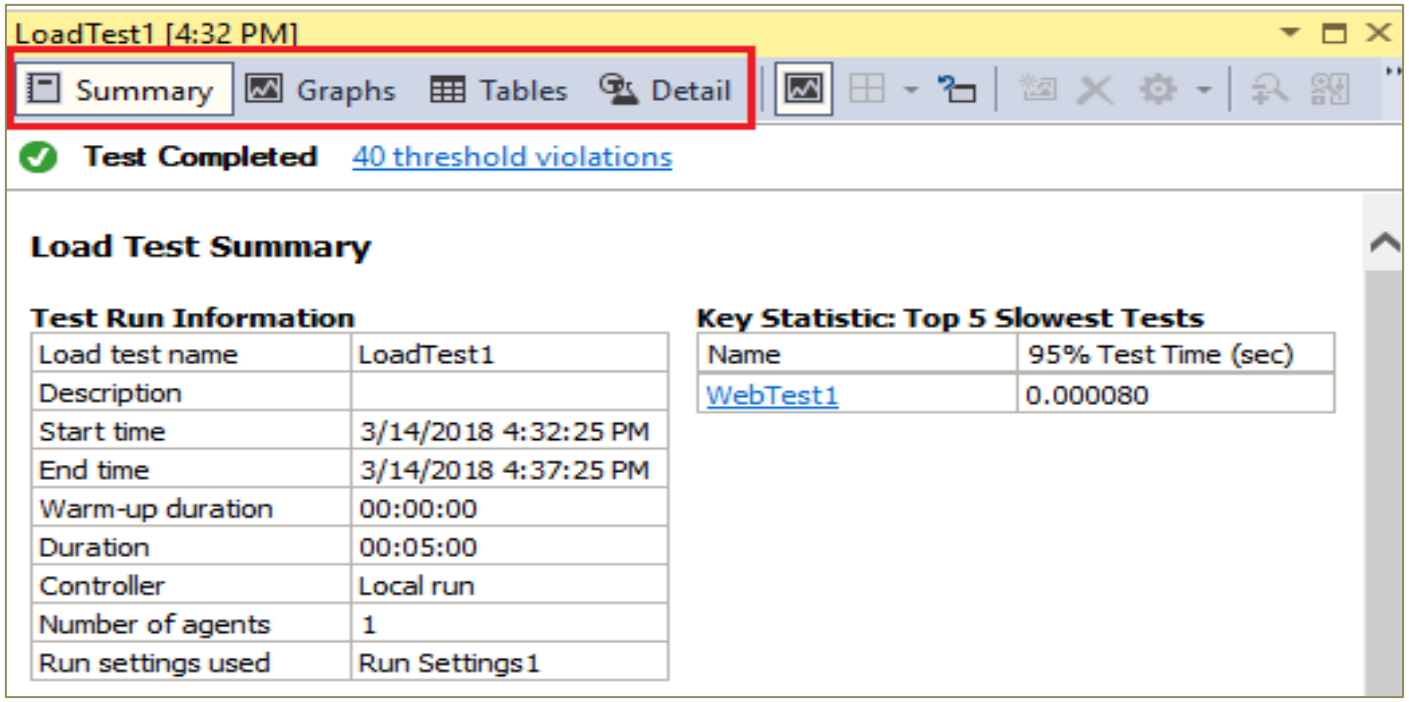
Inconvénients de sessions/cookies

2.Problème de la montée en charge

- **Outil de test de la montée en charge:**

-Il y’a un outil qui permet d’automatiser ce test en visual studio

«*Web performance and load testing tools* »

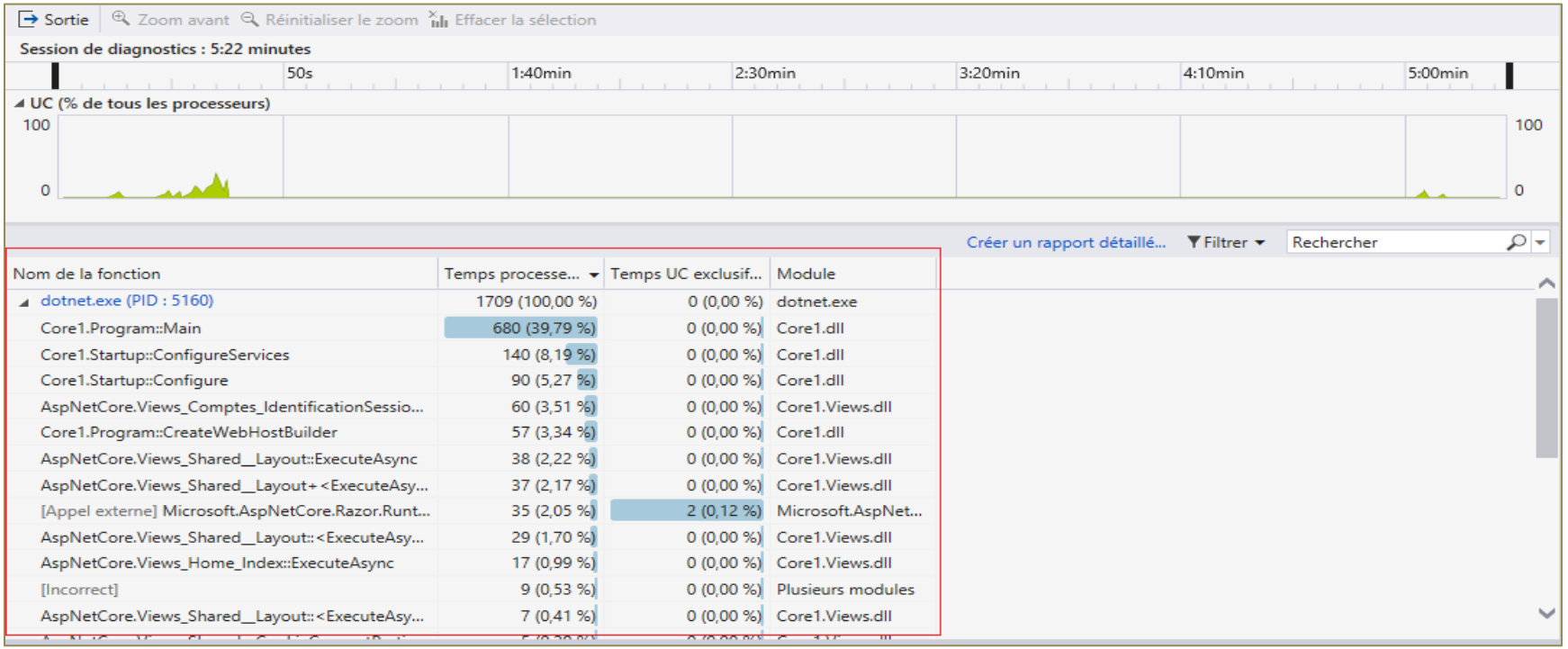


Inconvénients de sessions/cookies

2.Problème de la montée en charge

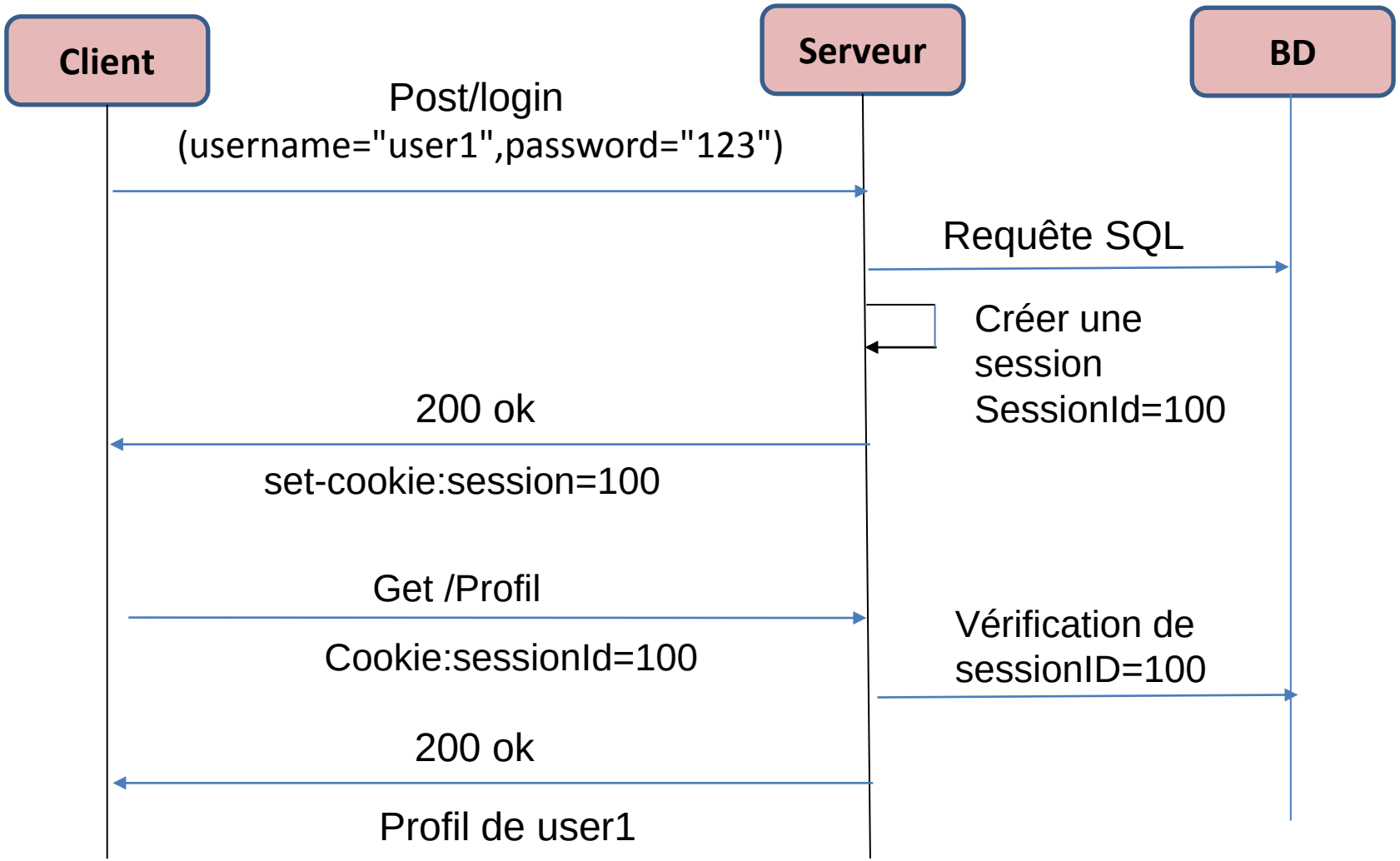
- **Outil de test de la montée en charge:**

VS2017 offre aussi un outil d'analyse de l'exécution appelé « **profiler** » ***Debug > Profiler > profileur de performance***



Inconvénients de sessions/cookies

2.Problème de la montée en charge



Inconvénients de sessions/cookies

-La montée en charge et les sessions:

-Lorsque on utilise une session,le serveur est trop chargé car à chaque fois un client envoie une demande,le serveur se charge de refaire un test sur ce client connecté en comparant l'id-session avec celui enregistré dans la base de donnée(autre requête dans le database) →saturation de la bande passante surtout si on a plusieurs clients connectés

Solution

Chercher un moyen qui permet de gérer la session coté client tout en garantissant la sécurité de ces données

JWT-Json Web Token-

-JWT est un standard ouvert (RFC 7519) définit une manière compacte et autonome de transmission sécurisée d'informations entre les parties sous la forme d'un objet JSON.

-Ce mode de transmission est bien sécurisé car les informations sont signées numériquement en Base64URL

JWT-Json Web Token-

-Compacte: puisque la taille est réduite donc il y'a la possibilité d'envoyer les informations d'une manière rapide dans l'entéte http,URL ou dans un paramètre POST

-Autonome: Le jwt contient toute les informations nécessaires sur l'utilisateur et par la suite le serveur ne sera pas obligé d'interroger avec d'autres tiers pour connaitre plus de détail sur l'utilisateur authentifié

JWT-Json Web Token-

- **Structure Jeton JWT:**

Un jeton JWT se compose de trois parties séparées par des points (.) :

<header> <payload> <signature>

Chacune de ces trois composantes est codé en Base64Url.

Structure JWT

•**JWT Header:**

-L'en-tête se compose généralement d'un objet JSON de deux membres :

- •Typ: le type du jeton, qui est donc JWT
- •Alg : l'algorithme de hachage utilisé:HMAC ou RSA.

```
{
 "typ": "JWT",
  "alg": "HS256"
}
```

- •Cet objet est encodé par la suite en Base64URL qui peut être transmis via l'URL

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Structure JWT

- **JWT Payload:**

-Payload contient les revendications (claims) sous le format JSON.

-Les claims sont des informations sur:

- une entité (généralement l'utilisateur)

- Des métadonnées supplémentaires.

-Il existe trois types de claims: reserved, public et private.

Structure JWT

- **JWT Payload:**

❑**Claims réservées:** Sont un ensembles des claims prédéfinies qui ne sont pas obligatoires mais recommandées.Exemple

- iss (issuer) :émetteur de token,celui qui va générer le token
- sub :sujet contient des informations sur l'utilisateur authentifié
- exp: heure d'expiration
- aud(audience) : audience pour laquelle ce jeton est destiné.
- lat (issued at) :date de création du token
- nbf(note before) :à ne pas utiliser avant cette date

Structure JWT

- **JWT Payload:**

❑**Claims publiques:** Elles peuvent être définies à volonté. Mais pour éviter les collisions, leurs noms devraient être choisis dans l'annuaire IANA JSON Web Token Registry

https://www.iana.org/assignments/jwt/jwt.xhtml

Structure JWT

- **JWT Payload:**

❑**Claims privées:** Ce sont des déclarations personnalisées créées pour partager des informations entre les parties qui ont prévu de les utiliser.

Exemple:

```
{
   "sub": "OUARRACHI",
   "iss": "http:test.jwt",
    "aud": "mobile app",
    "exp": "1485968105"
   }
```

Structure JWT

- **JWT Payload:**

❑**Exemple:**

```
{
   "sub": "OUARRACHI",
   "iss": "http:test.jwt",
    "aud": "mobile app",
    "exp": "1485968105"
   }
```

-Le payload encodé en Base64URL

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiJPVUFSUkFDSEkiLCJpc3MiOiJodHRwOnR
lc3Quand0IiwiYXVkIjoibW9iaWxlIGFwcCIsImV
4cCI6IjE0ODU5NjgxMDUifQ.QDJsp4NeBfDISIAA
svxEmEQDrBLfIC2fsYuafN-gSx4
```

Structure JWT

- **JWT signature:**

-La signature est utilisée pour vérifier que l'expéditeur du jeton JWT est bien celui qu'il prétend être et s'assurer que le message n'a pas été modifié en cours de route.

-Le JWT est signé en utilisant un secret (avec l'algorithme HMAC) ou une paire de clés publique/privée en utilisant RSA.

Structure JWT

- **JWT signature:**

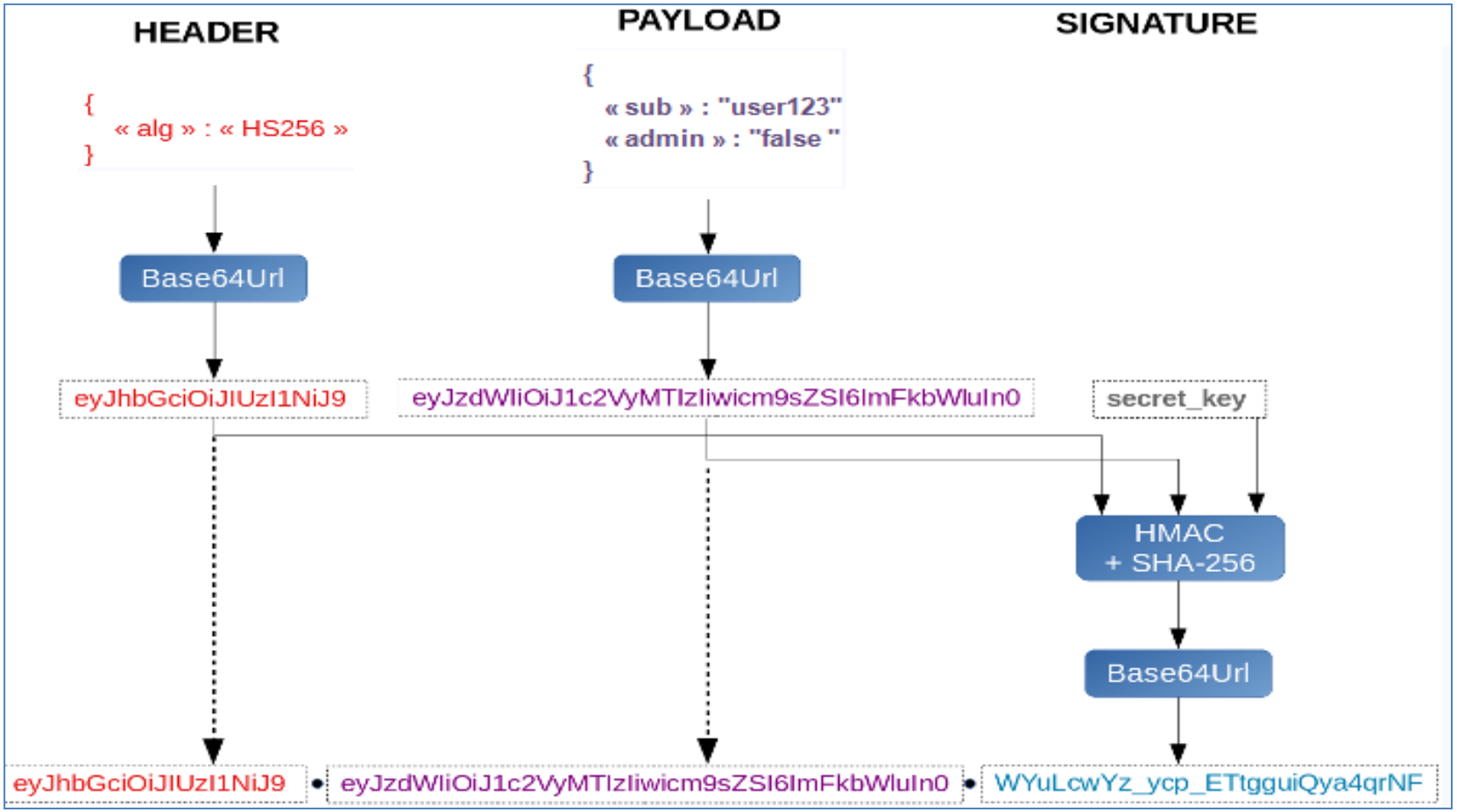
La signature est donc générée à partir de la concaténation des parties “Header” et “Payload” encodées en Base64, soit, par exemple, avec la clé privée “L2VE5VpgChrVPmgh1hgL” :

```
Signature = HMACSHA256(
   base64UrlEncode( {"alg":"HS256",
   "typ":"JWT"} ) +"." +
   base64UrlEncode( {"iat": 1480929282, "exp":
   1480932868, "name":"Username"} ),
   L2VE5VpgChrVPmgh1hgL
   )
```

Structure JWT

- **JWT signature:**

-Algorithme HMAC



Validité de JWT

- Si une personne modifie le contenu de ce jeton(header ou payload) la signature change radicalement. Et sans la connaissance du secret, cette personne est incapable de régénérer une signature valide.
- Par exemple, si une personne modifiait la claim "Admin" par "true", la signature actuelle ne serait plus correcte. Car en re-calculant côté serveur cette signature à partir de la clé secrète "*s3cr3t*" et l'algorithme *HMAC + SHA-256*, on devrait avoir pour ces données la valeur : *WYuLcwYz_ycp_ETtgguiQya4qrNF*
- C'est de cette manière que le système peut faire confiance au jeton JWT et s'assurer qu'il n'a pas été altéré par une personne extérieure au SI.

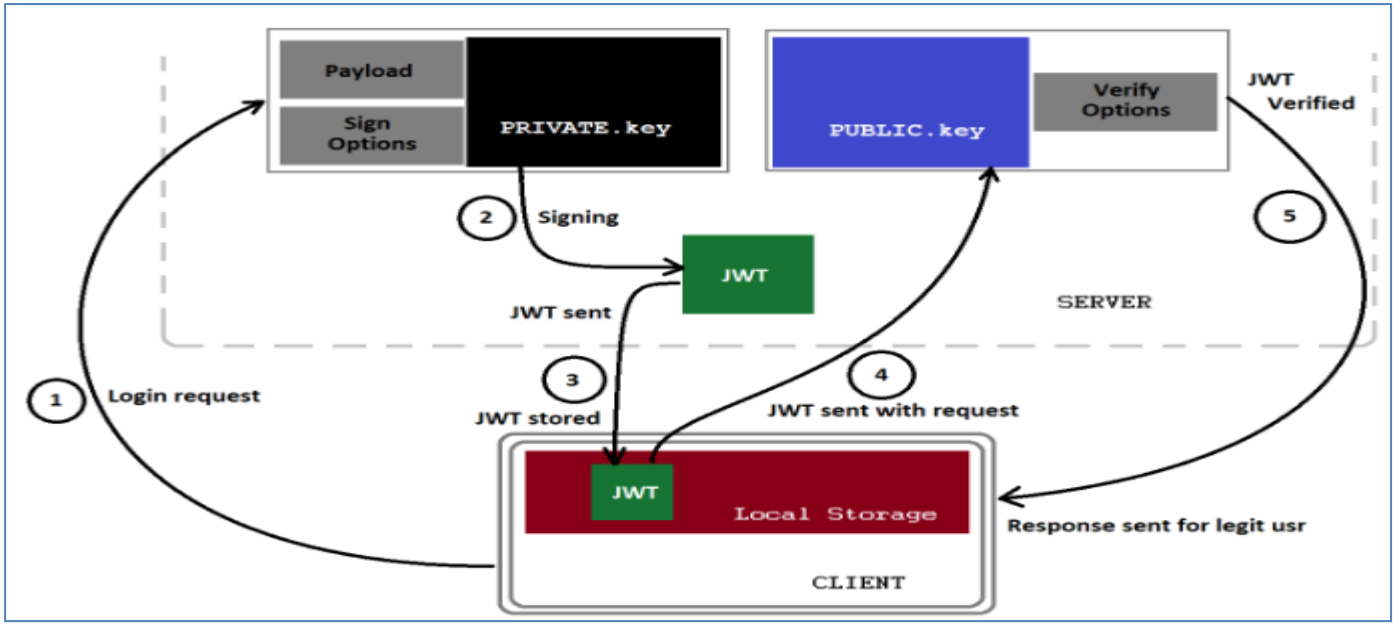
JWT

- Algorithme RSA

Le RSA utilise deux clés:

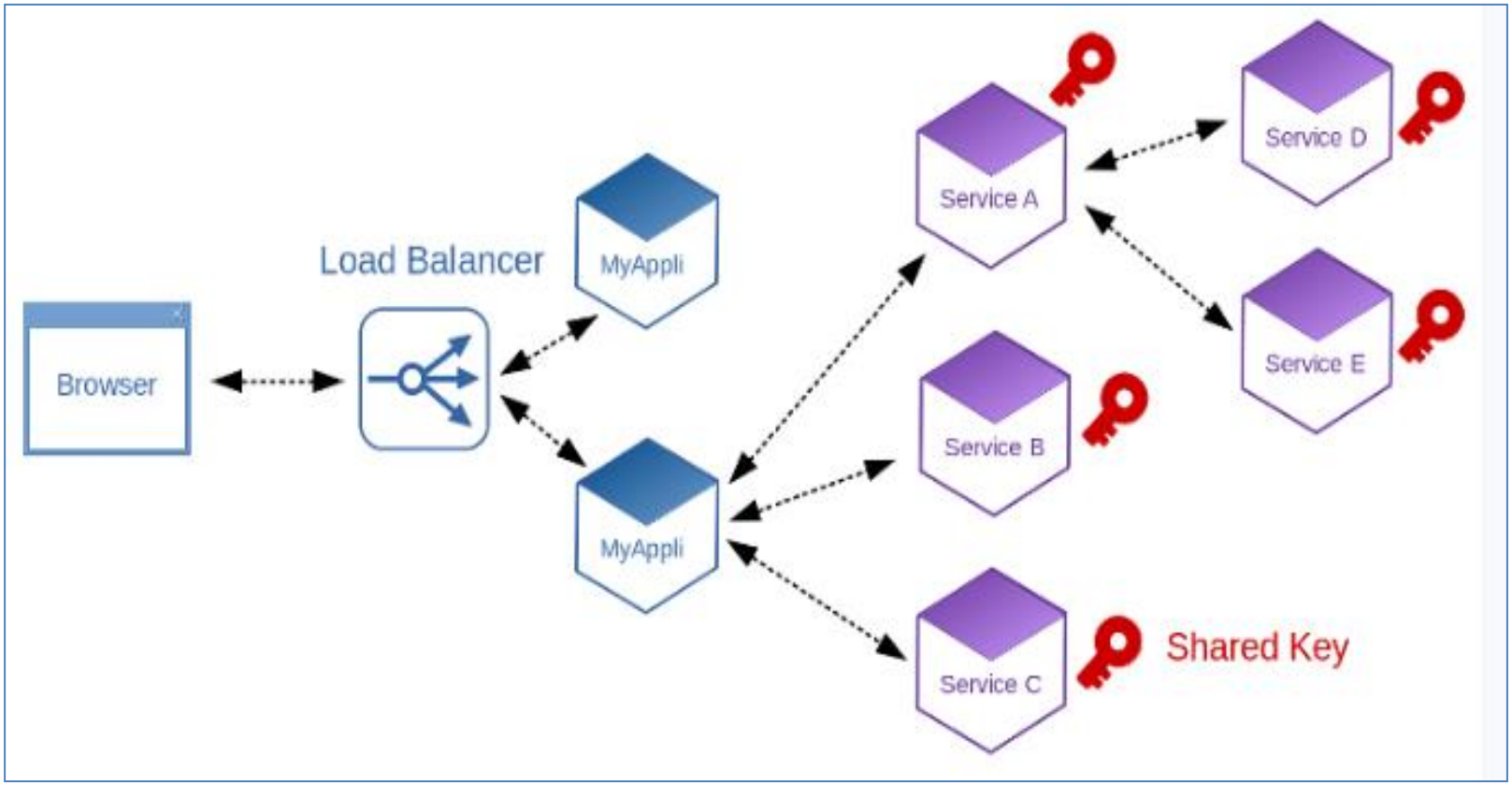
-une clé privée: pour générer le jwt

-Une clé publique: pour vérifié la validité des jwt



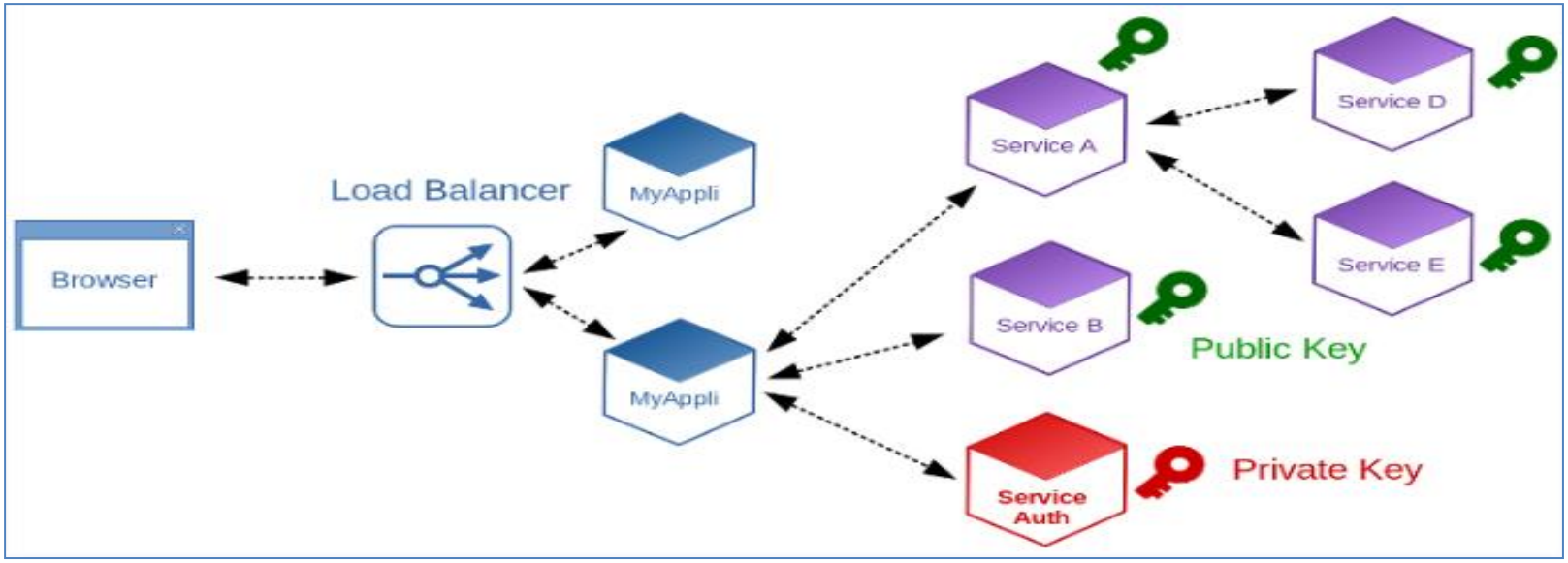
JWT et les microservices

- **Algorithme HMAC:**



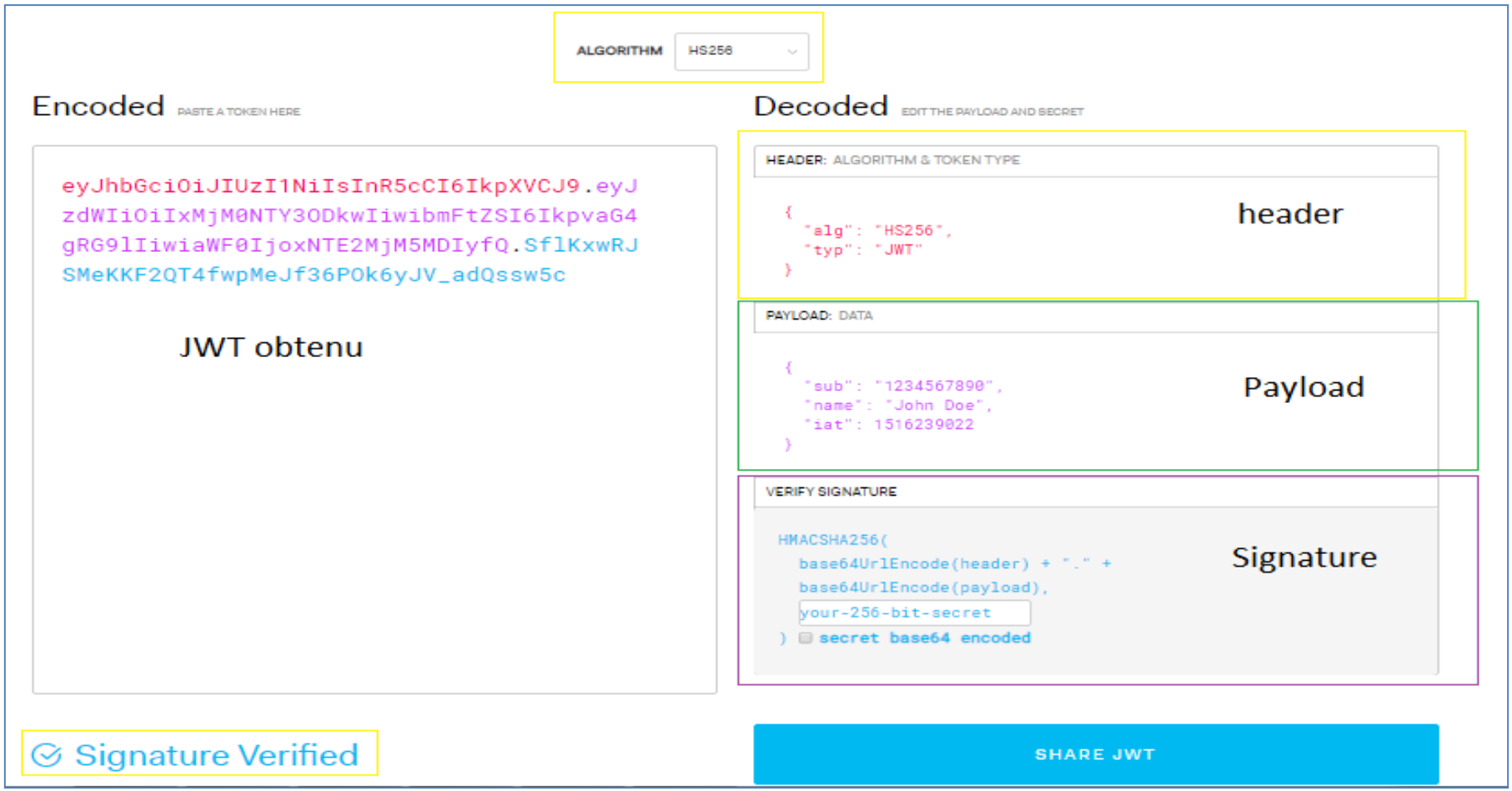
JWT et les microservices

- **Algorithme RSA:** Le RSA est plus sécurisé surtout si on travaille dans une architecture microservices car ce n'est pas la peine de distribuer la clé privé dans tous les serveurs.un seul qui va avoir cette clé pour construire le jeton



JWT

https://jwt.io/



JWT

- **Récapitulatif de fonctionnement de jwt :**

1. Au 1er temps le client demande une connexion

2. le serveur crée le jeton et l'envoit au navigateur

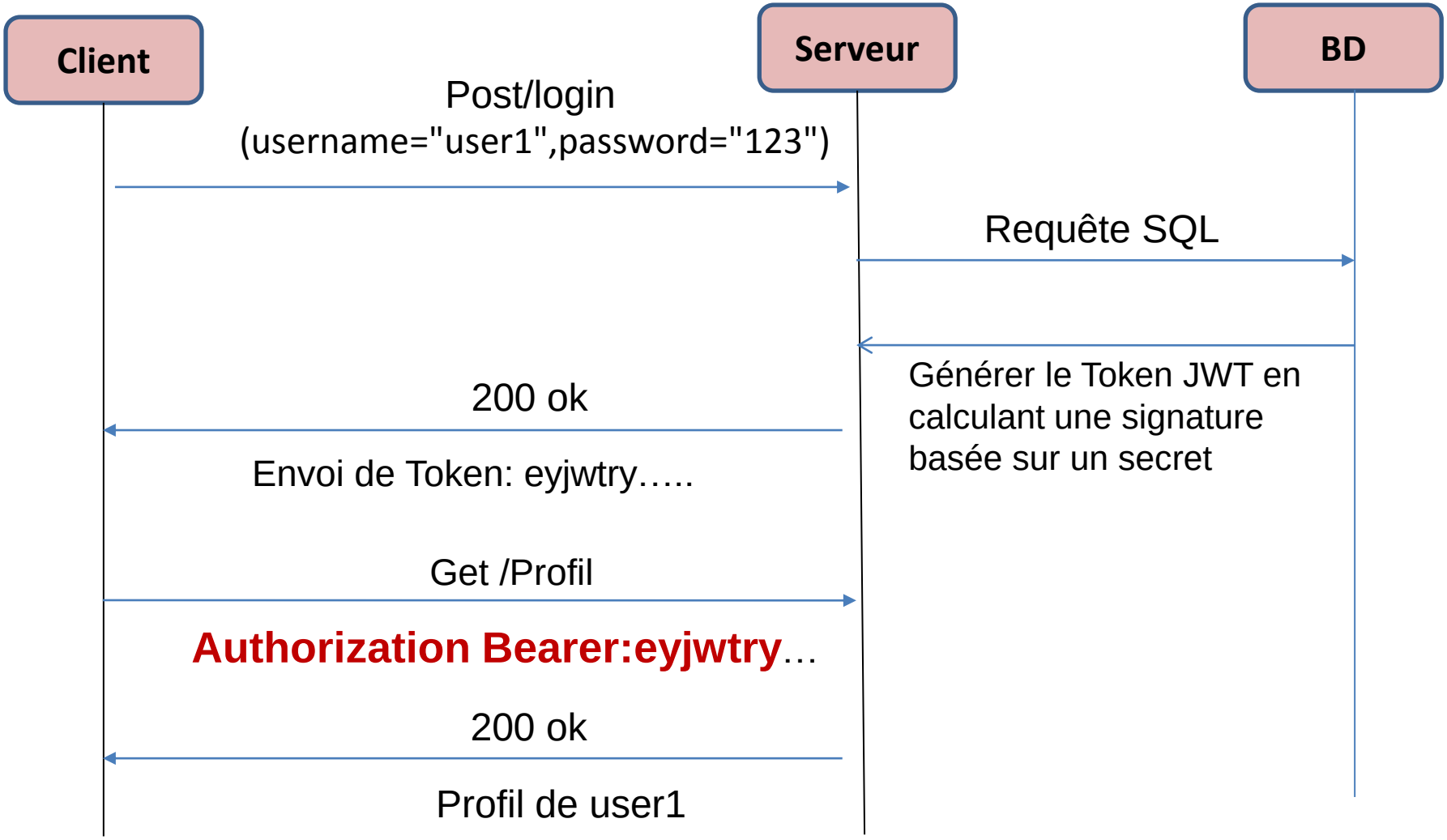
3. Par la suite ce jeton s'enregistre localement chez le client

4. Dans les prochaines connexions , l'utilisateur doit envoyer le jwt dans l'entéte Authorisation en utilisant le schéma Bearer:

Authorization:bearer <jeton>

JWT

- **Récapitulatif de fonctionnement de jwt :**



JWT

- **Problème de la montée en charge:**

On a pas ici ce problème car le token contient toutes les informations nécessaires sur l'utilisateur sans avoir besoin de passer par une application tierce pour extraire ces informations.

Implémentation de jwt en ASP.net core

Créer un projet ASP.Net core→API

Etape1:configuration de l'authentification basée sur JWT

Pour cela aller au fichier « startup » puis enregistrer un schéma d'authentification JWT en utilisant la méthode « AddAuthentication » .

Ce schéma consiste à fournir un ensemble de paramètres au serveur pour qu'il pourra créer le token.

Implémentation de jwt en ASP.net core

Etape1:configuration de l'authentification basée sur JWT

```
public void ConfigureServices(IServiceCollection services)
{
 services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
  .AddJwtBearer(options =>
  {
      var signingKey = Convert.FromBase64String(Configuration["Jwt:key"]);
      options.TokenValidationParameters = new TokenValidationParameters
      {   ValidateIssuer = false,
          ValidateAudience = false,
          ValidateLifetime = true,//Vérifier si le jeton n'a pas expiré
          ValidateIssuerSigningKey = true,//vérifier la clé de signature de l'émetteur est valide
          ValidIssuer = Configuration["Jwt:Issuer"], //définir la valeur de Issuer
          ValidAudience = Configuration["Jwt:Issuer"],  //définir la valeur de audience
          IssuerSigningKey = new SymmetricSecurityKey(signingKey)
      };
  });
```

Implémentation de jwt en ASP.net core

Etape1:configuration de l'authentification basée sur JWT

On définit les valeurs de l'émetteur, l'audience et la clé de signature dans le fichier « appsetting.json »

```
"Jwt": {
  "Key": "tKE+pMd2rQAHBbOjXWTZqacLJRLq1rnTzZdmKRJEXLjtiGOnFY3w+vuUxPSgLdMFbbVXxPrFWNUd/yQyG5PsEg==",
  "Issuer": "Test.com",
  "ExpiryDuration": 10
},
```

Implémentation de jwt en ASP.net core

Etape1:configuration de l'authentification basée sur JWT

Ensuite il faudra rendre le service d'authentification disponible pour l'application

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseAuthentication();
    app.UseMvc();
}
```

Implémentation de jwt en ASP.net core

Etape2:Générer le token JWT

Pour cela on crée un controller dans lequel on définit les claims nécessaires qui vont permettre au serveur de générer le token.

- Model:User

```
public class User
{
    public int Id { get; set; }

    public string Username { get; set; }

    public string Password { get; set; }

    public string Role { get; set; }
}
```

Implémentation de jwt en ASP.net core

Etape2:Générer le token JWT

- **Controller:Login**

```
[Route("api/[controller]")]
[ApiController]
public class LoginController : ControllerBase
{

    private readonly IConfiguration _configuration;

    public LoginController(IConfiguration config)
    {
        _configuration = config;
    }

    private readonly IEnumerable<User> _users = new List<User>
    {
        new User { Id = 1, Username = "meryem", Password = "123", Role = "Administrator"},
        new User { Id = 2, Username = "malak", Password = "456", Role = "invité"},

    };
```

Implémentation de jwt en ASP.net core

Etape2:Générer le token JWT

```
private string GenerateJSONWebToken(User userInfo)
{  var user = _users.Where(x => x.Username == userInfo.Username && x.Password == userInfo.Password).SingleOrDefault();
    if (user == null)
    {  return null;   }
    var signingKey = Convert.FromBase64String(_configuration["Jwt:Key"]);
    var expiryDuration = int.Parse(_configuration["Jwt:ExpiryDuration"]);
    var tokenDescriptor = new SecurityTokenDescriptor
    {  Issuer = null,               // Not required as no third-party is involved
       Audience = null,             // Not required as no third-party is involved
       IssuedAt = DateTime.UtcNow,
       NotBefore = DateTime.UtcNow,
       Expires = DateTime.UtcNow.AddMinutes(expiryDuration),
       Subject = new ClaimsIdentity(new List<Claim> {
              new Claim("userid", user.Id.ToString()),
              new Claim("roles", user.Role),
              new Claim("Usernames", user.Username.ToString()),
          }),
SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(signingKey), SecurityAlgorithms.HmacSha256Signature)
    };
    var jwtTokenHandler = new JwtSecurityTokenHandler();
    var jwtToken = jwtTokenHandler.CreateJwtSecurityToken(tokenDescriptor);
    var token = jwtTokenHandler.WriteToken(jwtToken);
    return token;  }
```

1.Définir les paramètres de token(claims,clé de cryptage)

2.Créer le token

Implémentation de jwt en ASP.net core

Etape2:Générer le token JWT

- Méthode d'authentification

```
[HttpPost]
public IActionResult Login2([FromBody]User user)
{
    var jwtToken = GenerateJSONWebToken(user);

    if (jwtToken == null)
    {
        return Unauthorized();
    }
    return Ok(jwtToken);
}
```

Implémentation de jwt en ASP.net core

Etape3:Consommer le token JWT

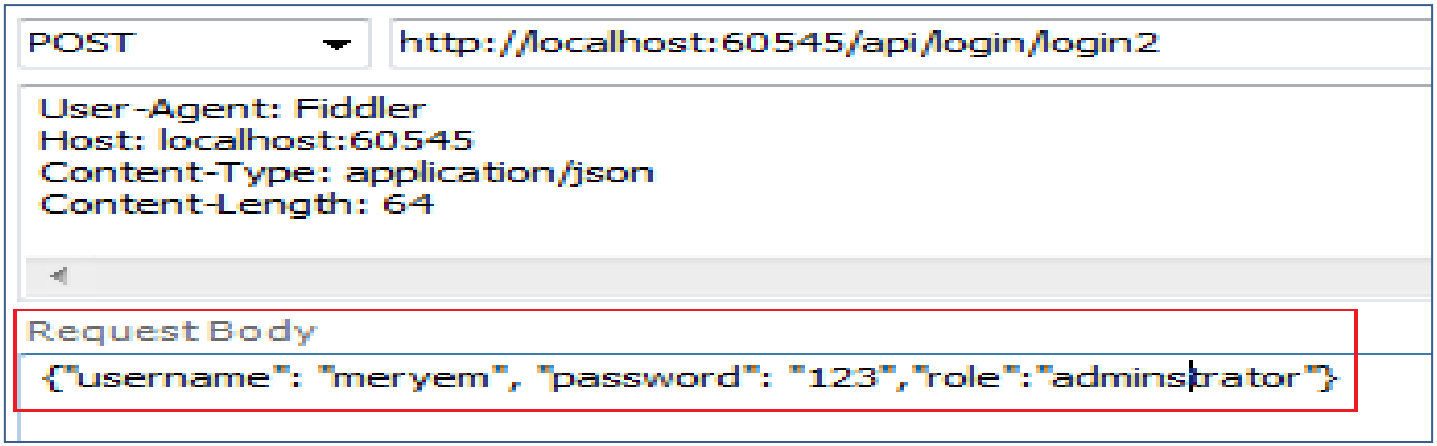
- **Récupération de L'utilisateur courant:**

```
[HttpGet]
[Authorize]
public IActionResult Get()
{   var currentUser = HttpContext.User.Claims.Where(x => x.Type == "userid").SingleOrDefault();
    return Ok( $"utilisateur connecte a un id"+ currentUser.Value);
    /******************Ou bien ******************************
    List<Claim> userClaimsList = HttpContext.User.Claims.ToList();
    string role = userClaimsList[1].ToString();
    string nom = userClaimsList[2].ToString();
    return Ok($"utilisateur connecte a comme role=" + role +"et nom="+ nom);*/
}
```

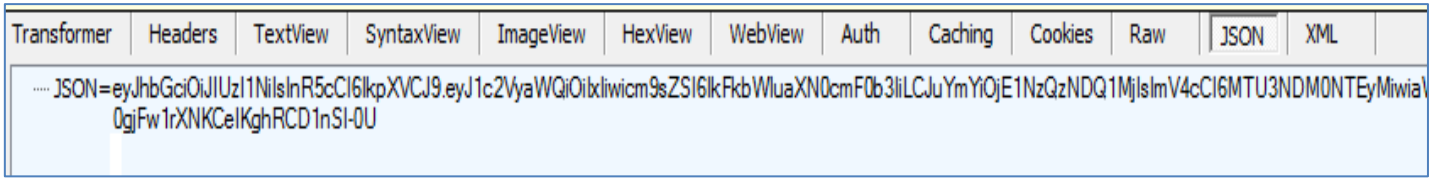
Implémentation de jwt en ASP.net core

- **Tester le token sous fiddler:**

  - **Méthode login**



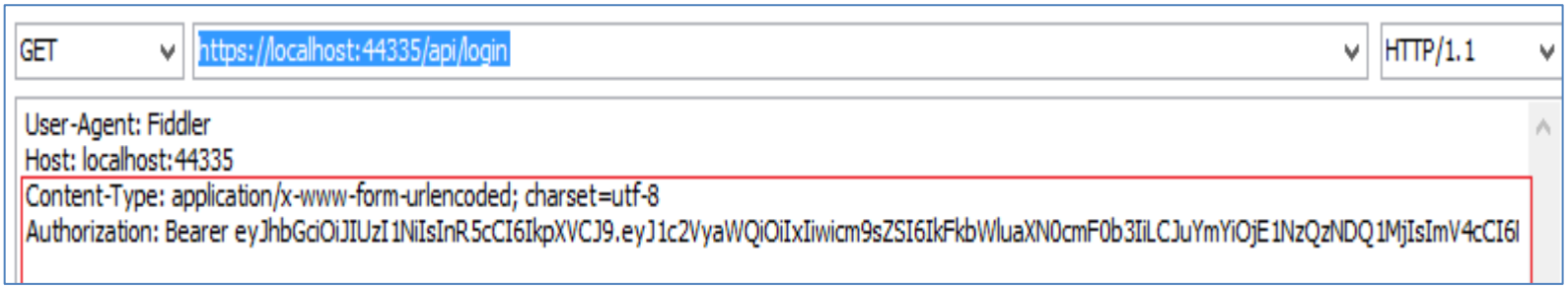
-Résultat:



Implémentation de jwt en ASP.net core

- Tester le token sous fiddler

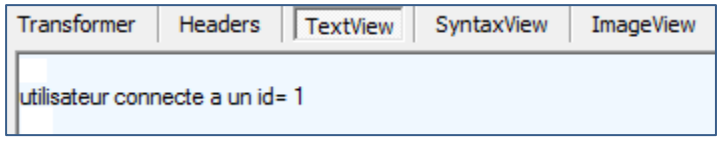
  - **Méthode Get**



Content-Type: application/x-www-form-urlencoded; charset=utf-8

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5……

-Résultat



Implémentation de jwt en ASP.net core

❑ Stockage de jwt:

- **Local Storage:** est un moyen de stocker des données sur le navigateur en utilisant javascript.Il offre une capacité de stockage beaucoup plus grande.

```
localStorage.setItem('jwtToken', 'user1');
var x = localStorage.getItem('jwtToken');
```

-*Type d'application:* Application SPA ou mobile

-*Sécurité:* sensible aux attaques XSS

Implémentation de jwt en ASP.net core

❑ Stockage de jwt:

- **Session:** on l'utilise dans le cas des applications SPA, et web

-*Sécurité:* sensible aux attaques XSS

- **Cookies:** applications web,applications SPA

-*Sécurité:* sensible aux attaques CSRF (httponly=true et secure)

Implémentation de jwt en ASP.net core

❑ Stockage de jwt:

Remarque: N'oublier pas de définir des en-têtes d'autorisation après la connexion de l'utilisateur,car pour chaque requête ultérieure,le serveur attend le préfixe Authorization Bearer

-Pour les projets ASP.Net core:Dans la méthode de configuration de startup

```
app.Use(async (context, next) =>
{
    var JWToken = context.Request.Cookies["JWToken2"];
    if (!string.IsNullOrEmpty(JWToken))
    {
        context.Request.Headers.Add("Authorization", "Bearer " + JWToken);
    }
    await next();
});
```

JWT

❑ Inconvénients de JWT

- Le principal inconvénient des JWT est qu'ils ne peuvent pas être révoqués ; en effet, une fois émis, il est possible de valider le token sans appel à la base de données ou au service qui l'aurait émis. En cas de compromission du token, il faut attendre que celui-ci expire .
- Solution: créer une liste de tokens invalidés dans un cache qui sera consulté avant d'effectuer la validation d'un token (ou l'inverse, qui consiste à garder la liste des tokens valides)

Sécurité

Attaque CSRF

-En ASP.NET Core2,0, on ajoute l'attribut [ValidateAntiForgeryToken] pour se protéger contre les attaques CSRF.

```
public class ManageController
{
   [HttpPost]
   [ValidateAntiForgeryToken]
   public IActionResult ChangePassword()
   {
      // ...
      return View();
   }
}
```

Attaque CSRF

-Attribut [AutoValidateAntiForgeryToken] fonctionne de manière identique à l'attribut [ValidateAntiForgeryToken], sauf qu'il ignore les méthodes "sûres" telles que GET .

-L'ajout de l'attribut à votre application est simple: ajoutez-le simplement à la collection de filtres globaux de votre classe Startup lorsque vous appelez AddMvc ().

```
public class Startup
{
  public void ConfigureServices(IServiceCollection services)
  {
    services.AddMvc(options =>
    {
        options.Filters.Add(new AutoValidateAntiforgeryTokenAttribute());
    });
  }
}
```

```
[AutoValidateAntiforgeryToken]
public class AntiForgeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
    [HttpPost]
    public IActionResult Index(string userName)
    {
        return View("Index", userName);
    }
    [HttpDelete]
    public IActionResult RemoveUser(string userName)
    {
        string url = string.Format("~/RemovedUser/{0}", userName);
        return RedirectToAction("Account", "RemoveUser", "User");
    }
}
```

Asp.net core identity

- Fonctionne de la même façon que Forms Authentification en ASP MVC