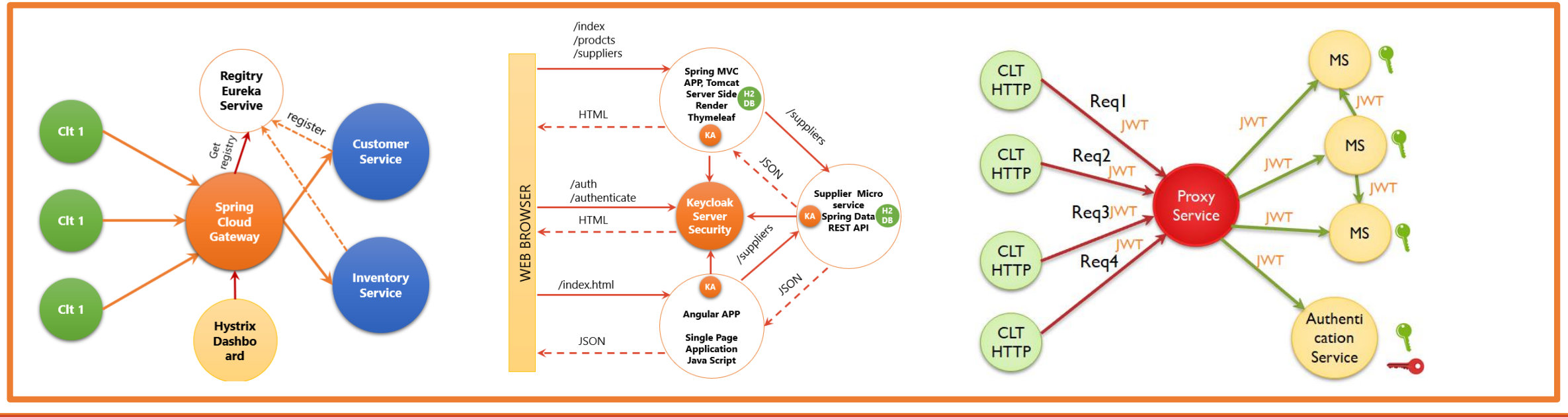
Sécurité des application Web Mobiles et Systèmes Distribués Micro services



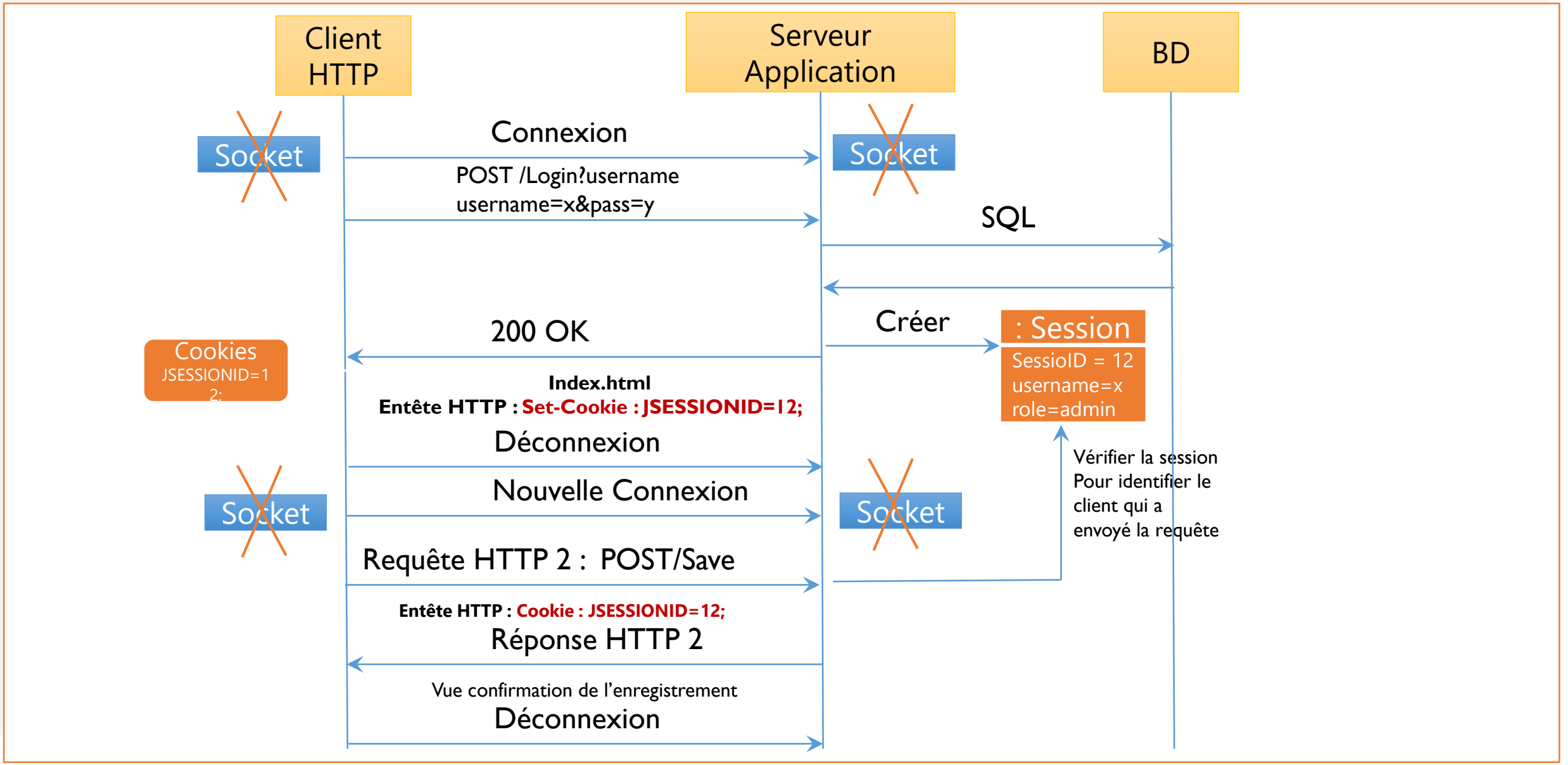
Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Systèmes d'authentification

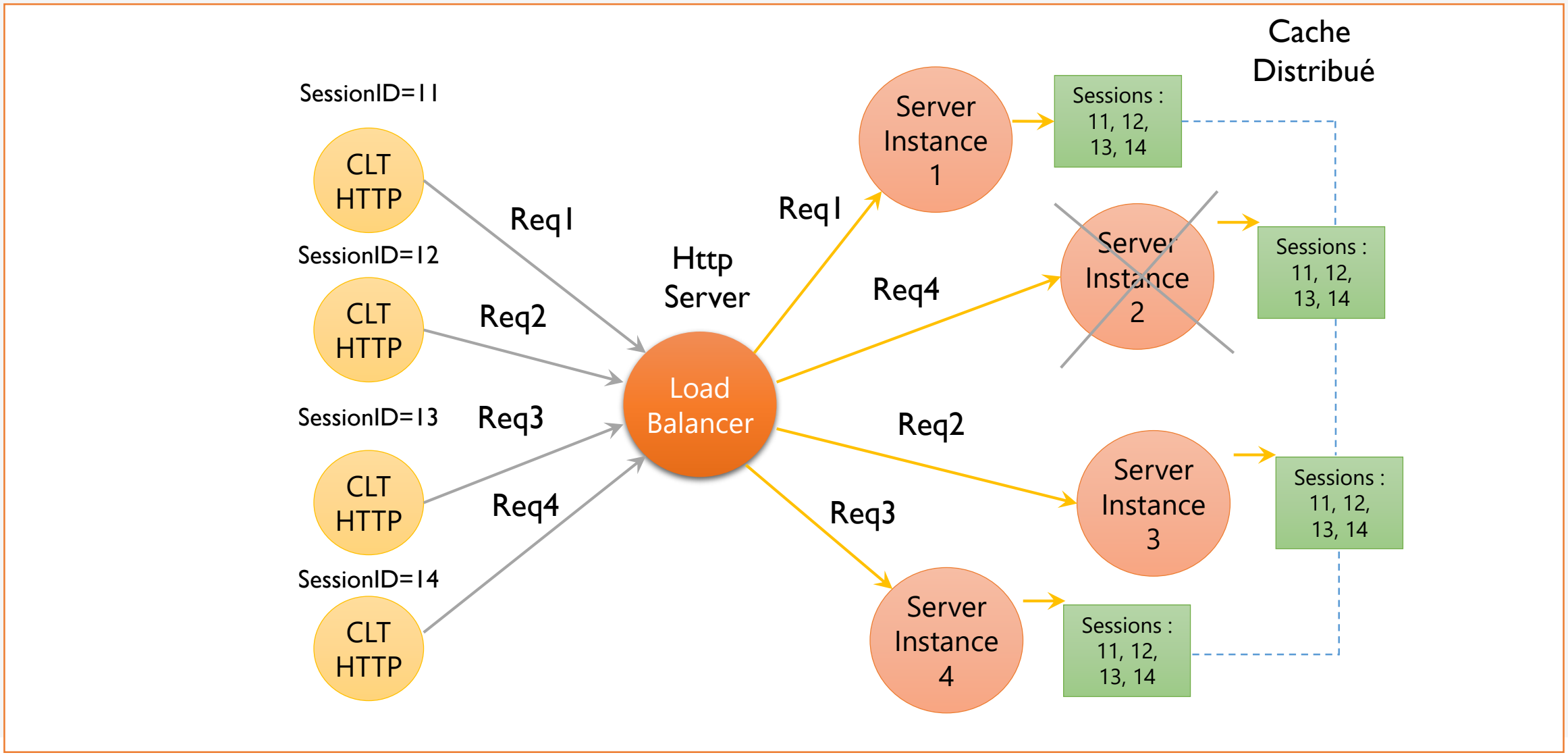
Deux types de modèles d'authentification :

- **Statful** : Les données de la session sont enregistrés coté serveur d'authentification
- **Statless** : les données de la session sont enregistrés dans un jeton d'authentification délivré au client.

Authentification Statful basée sur les Sessions et Cookies



Problème de montée en charge : Nécessité d'un Cache Mémoire partagé ou Distribué pour les sessions



Type d’attaque : Cross Site Request Forgery (CSRF)

- En sécurité informatique, le ***Cross-Site Request Forgery***, abrégé **CSRF** est un type de vulnérabilité des services d'authentification web.
- L’objet de cette attaque est de transmettre à un utilisateur authentifié
  - Une requête HTTP falsifiée qui pointe sur une action interne au site,
  - Afin qu'il l'exécute sans en avoir conscience et en utilisant ses propres droits.
  - L’utilisateur devient donc complice d’une attaque sans même s'en rendre compte.
  - L'attaque étant actionnée par l'utilisateur, un grand nombre de systèmes d'authentification sont contournés.

Cross Site Request Forgery : CSRF

- Exemple d'attaque CSRF :
  - Yassine possède un compte bancaire d'une banque qui lui donne entre autres la possibilité d'effectuer en ligne des virements de son compte vers d'autres comptes bancaires.
  - L'application de la banque utilise un système d'authentification basé uniquement sur les sessions dont les SessionID sont stockées dans les cookies.
  - Sanaa possède également un compte dans la même banque. Elle connait facilement la structure du formulaire qui permet d'effectuer les virements.
  - Sanaa envoie à Yassine, un email contenant un message présentant un lien hypertexte demandant Yassine de cliquer sur ce lien pour découvrir son cadeau d'anniversaire.
  - Ce message contient également un formulaire invisible permettant de soumettre les données pour effectuer un virement. Ce formulaire contient entre autres des champs cachés :
    - Le montant du virement à effectuer vers
    - Le numéro de compte de Sanaa
  - Au moment ou Yassine reçoie son message, la session de son compte bancaire est ouverte. Son Session ID est bien présent dans les cookies.
  - En cliquant sur le lien de l'email, Yassine, vient d'effectuer un virement de son compte vers celui de Sanaa, sans qu'il le sache.
  - En effet, en cliquant sur le lien, les données du formulaire caché sont envoyées au bon script coté serveur, et comme les cookies sont systématiquement envoyées au serveur du même domaine, le serveur autorise cette opération car, pour lui, cette requête vient d'un utilisateur bien authentifié ayant une session valide.

Préventions des attaques CSRF

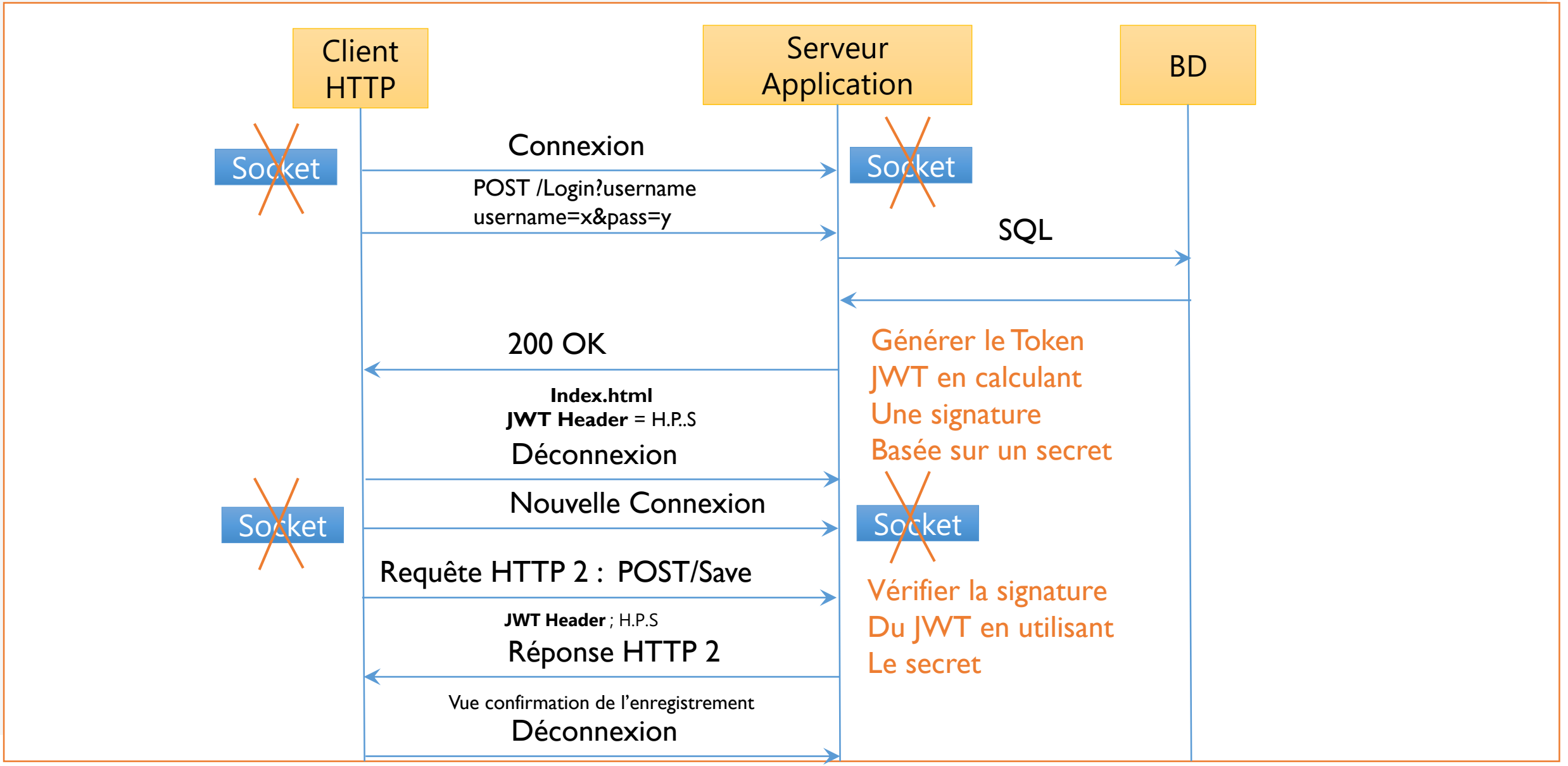
Utiliser des jetons de validité (**CSRF Synchronizer Token**) dans les formulaires :

- Faire en sorte qu'un formulaire posté ne soit accepté que s'il a été produit quelques minutes auparavant. Le jeton de validité en sera la preuve.
- Le jeton de validité doit être transmis souvent en paramètre (Dans un champs de type Hidden du formulaire) et vérifié côté serveur.

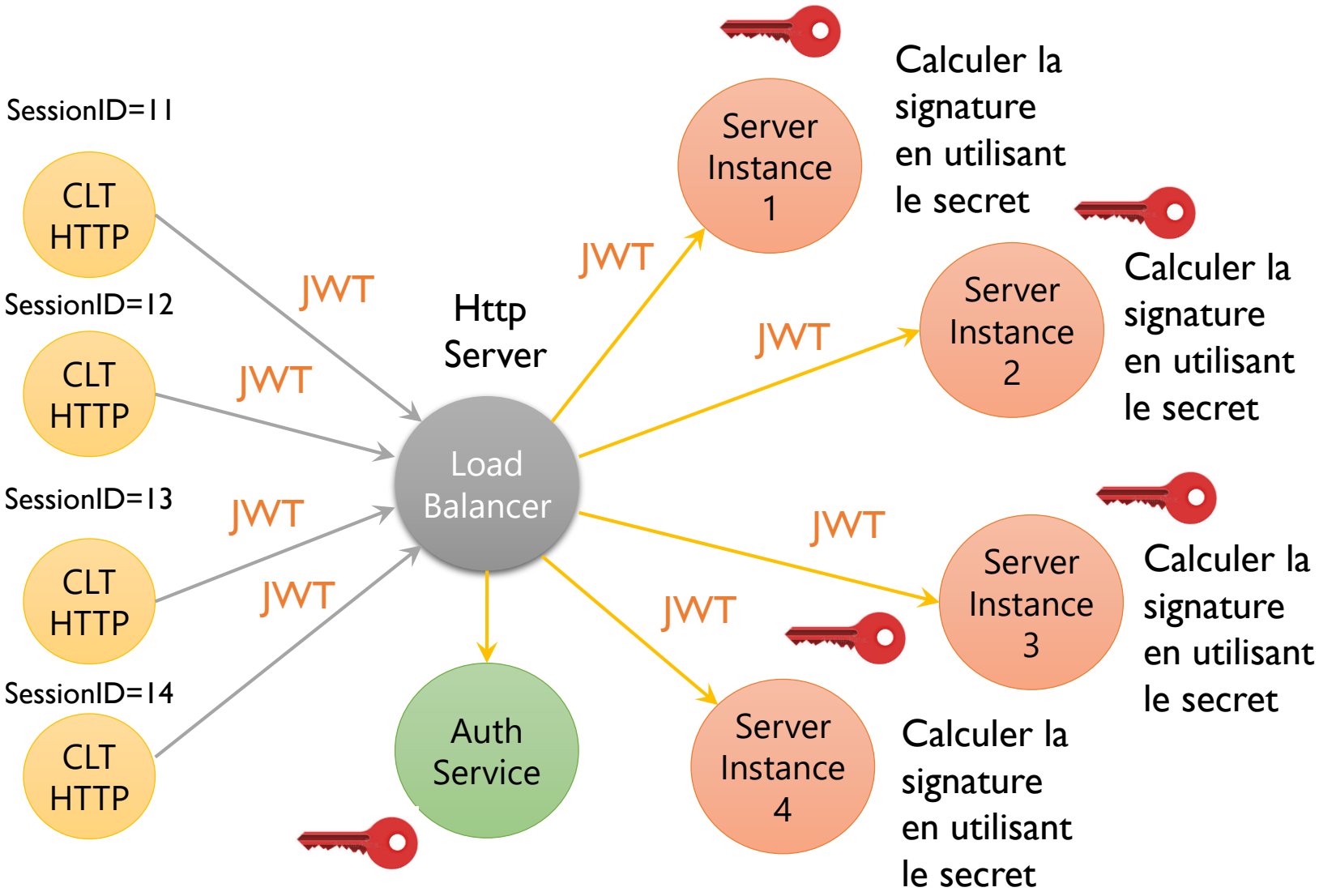
Autres présentions :

- Éviter d'utiliser des requêtes HTTP GET pour effectuer des actions critiques de modification des données (Ajout, Mise à jour, Suppression) : cette technique va naturellement éliminer des attaques simples basées sur les liens hypertexte, mais laissera passer les attaques fondées sur JavaScript, lesquelles sont capables très simplement de lancer des requêtes HTTP POST.
- Demander des confirmations à l'utilisateur pour les actions critiques, au risque d'alourdir l'enchaînement des formulaires.
- Demander une confirmation de l'ancien mot de passe à l'utilisateur pour changer celui-ci ou utiliser une confirmation par email ou par SMS.
- Effectuer une vérification du référent dans les pages sensibles : connaître la provenance du client permet de sécuriser ce genre d'attaques. Ceci consiste à bloquer la requête du client si la valeur de son référent est différente de la page d'où il doit théoriquement provenir.

Authentification Statless basée sur les Sessions et Cookies



Authentification Statless:
Problème de montée en charge : Pas de besoin de cache partagé ou distribué



Json Web Token (JWT)

- JSON Web Token (JWT) est un standard (RFC 7519) qui définit une solution **compacte** et **autonome** pour transmettre de manière sécurisée des informations entre les applications en tant qu'objet structuré au format JSON (Java Script Object Notation).
  - Compact : en raison de leur petite taille, les JWT peuvent être envoyés via une URL, un paramètre POST ou dans un en-tête HTTP. De plus, la plus petite taille signifie que la transmission est rapide.
  - Autonome : Le JWT contient toutes les informations requises sur l'utilisateur, ce qui évite d'avoir à interroger la base de données plus d'une fois pour connaitre le détail de l'identité d'un client authentifié.
- Le JWT est fiable car il est signé numériquement.
- JWT est constitué de trois parties séparées par un point « . » :
  - Header
  - Payload
  - Signature

La forme d'un JWT est donc :

- xxx.yyy.zzz

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgwL2F1dGgiLCJhdWQiOlsiV2ViIEZyb250IEVuZCIsIk1vYmlsZSBBcHAiXSwiZXhwIjo1NDc4OTAwNSwibmJmIjpudWxsLCJpYXQiOjQ5ODY1NDMyLCJqdGkiOiJpZHI1NjU0M2Z0dTg5MDk4NzYiLCJuYW1lIjoibWVkIiwicm9sZXMiOlsiYWRtaW4iLCJhdXRob3IiXX0.-V4FXAPpIBx1HqONU6qp7ptqzq32RrolDlhj4cVUQV0
```

Header

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

Encodage Base 64 URL

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Payload

```
{
  "sub": "1234567890",
  "iss":"http://localhost:8080/auth",
  "aud":["Web Front End","Mobile App"],
  "exp":54789005,
  "nbf":null,
  "iat":49865432,
  "jti":"idr56543ftu8909876",
  "name": "med",
  "roles":["admin","author"]
}
```

Encodage Base 64 URL

eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiQmFja2VuZCIsImF1ZCI6WyJXZWIg ....
vciJdfQ

Signature : RSA (2 clés publiques et privée) ou HMAC (1 clé privée)

HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

V4FXAPpIBx1HqONU6qp7ptqzq32RrolDlhj4cVUQV0

JWT : Header

- L'en-tête se compose généralement de deux parties:
  - Le type du jeton, qui est JWT,
  - L'algorithme de hachage utilisé, tel que :
    - **HMAC** (**H**ash **M**essage **A**uthentication **C**ode) : Symétrique (Clé privé)
    - **RSA** (**R**onald Rivest (2015), Adi **S**hamir (2013) et Leonard **A**dleman (2010).) : Asymétrique avec clés Publique et Clé Privée
- La structure du Header est un objet JSON ayant la forme la suivante :

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

- Cet objet JSON est ensuite encodé en Base64URL. Ce qui donne la forme suivante :

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

JWT : Payload

La deuxième partie du jeton est le Payload, qui contient les claims (revendications.)

Les claims sont des déclarations concernant :

- Une entité (généralement l'utilisateur)
- Des métadonnées supplémentaires.

Il existe trois types de claims :

- Enregistrées (Registred)
- publiques (Public)
- Privées (Private)

JWT : Payload

Registred Claims :

- Il s'agit d'un ensemble de revendications prédéfinies qui ne sont pas obligatoires mais recommandées pour fournir un ensemble de revendications utiles et interopérables. Certains d'entre eux sont:
  - iss (issuer : Origine du token),
  - exp (heure d'expiration),
  - sub (sujet),
  - aud (public cible),
  - nbf (Not Before : A ne pas utiliser avant cette date)
  - iat (issued at : date de création du token).
  - jti (JWT ID identifiant unique du JWT).

JWT : Payload

Public Claims :

- Celles-ci peuvent être définies à volonté par ceux qui utilisent des JWT.
- Mais pour éviter les collisions, elles doivent être définies dans le registre des jetons Web IANA JSON (IANA JSON Web Token Registry : https://www.iana.org/assignments/jwt/jwt.xhtml) ou être définies comme des URI contenant un espace de noms pour éviter les confusions.

JWT : Payload

Private Claims :

- Il s'agit des claims personnalisés créées pour partager des informations entre des parties qui acceptent de les utiliser et ne sont ni des réclamations enregistrées ni des réclamations publiques.

JWT : Exemple de Payload

```
{
  "sub": "1234567890",
  "iss":"Backend",
  "aud":["Web Front End","Mobile App"],
  "exp":54789005,
  "nbf":null,
  "iat":49865432,
  "jti":"idr56543ftu8909876",
  "name": "med",
  "roles":["admin","author"]
}
```

Le payload est encodé en Base64URL. Ce qui donne :

eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiQmFja2VuZCIsImF1ZCI6WyJXZWIgRnJvbnQgRW5kIiwiTW9iaWxlIEFwcCJdLCJleHAiOjU0Nzg5MDA1LCJuYmYiOm51bGwsImlhdCI6NDk4NjU0MzIsImp0aSI6ImlkcjU2NTQzZnR1ODkwOTg3NiIsIm5hbWUiOiJtZWQiLCJyb2xlcyI6WyJhZG1pbiIsImF1dGhvciJdfQ

JWT : Signature

La signature est utilisée pour :

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

Par exemple si vous voulez utiliser l'algorithme HMAC SHA256, la signature sera créée de la façon suivante:

- HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

jwt.io

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

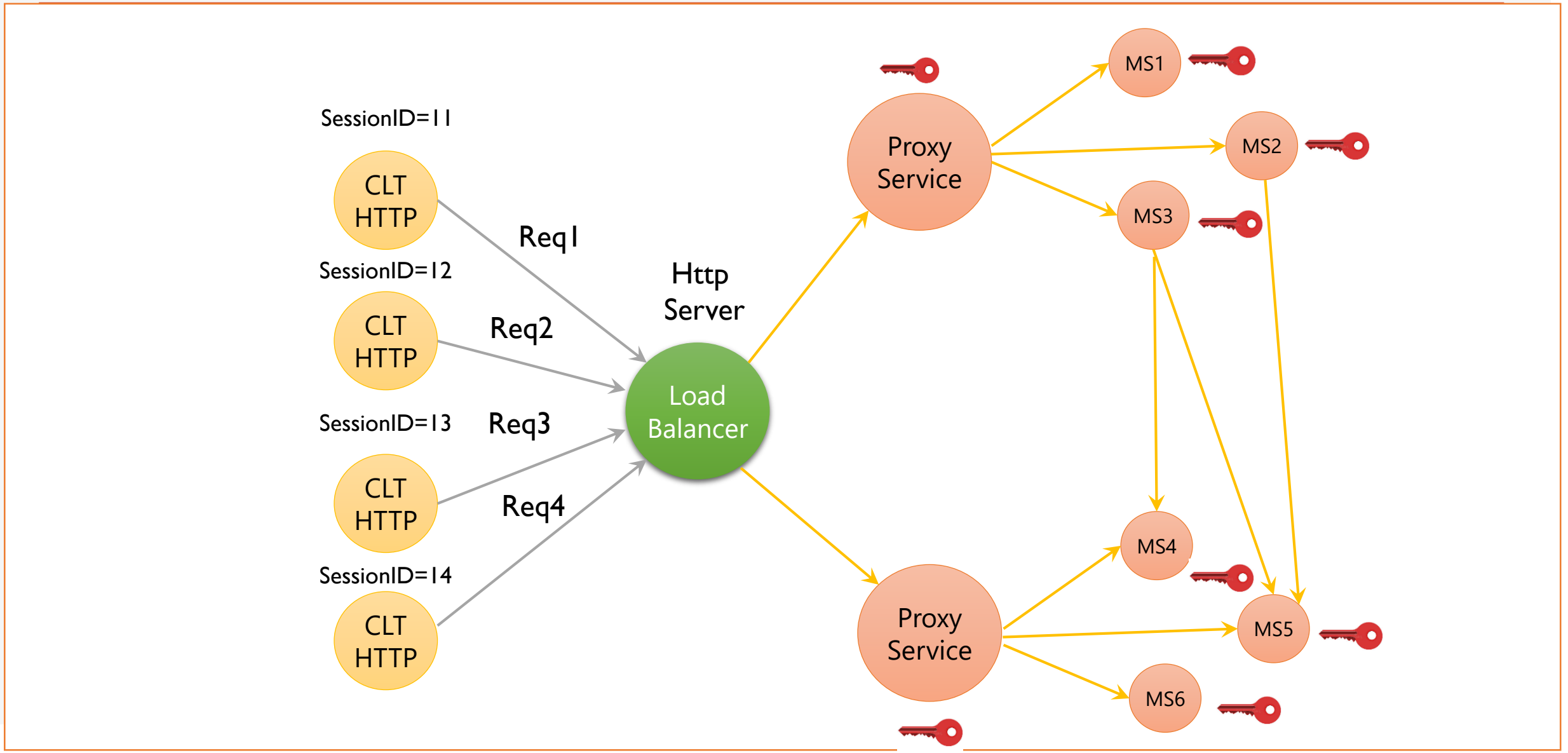
PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

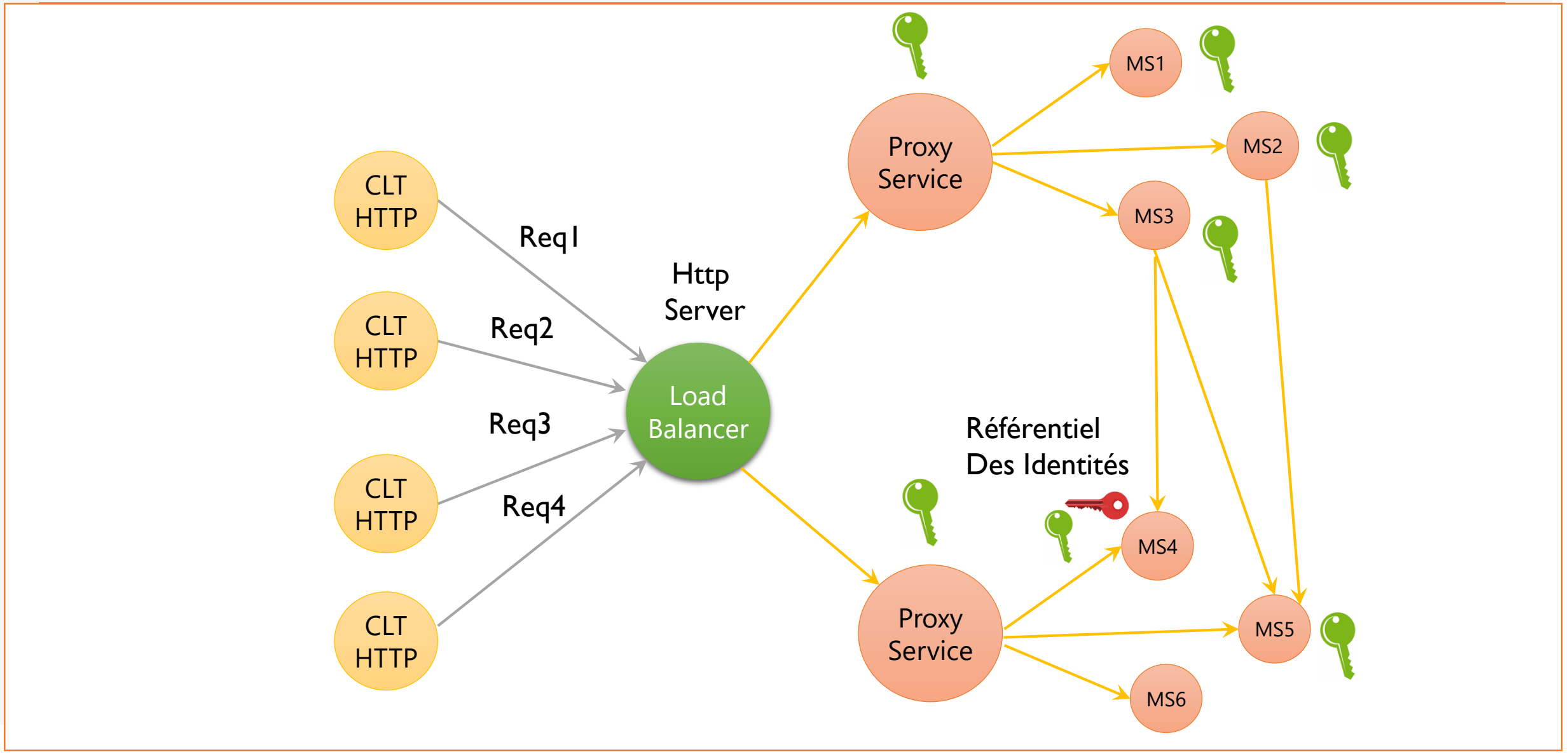
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
```

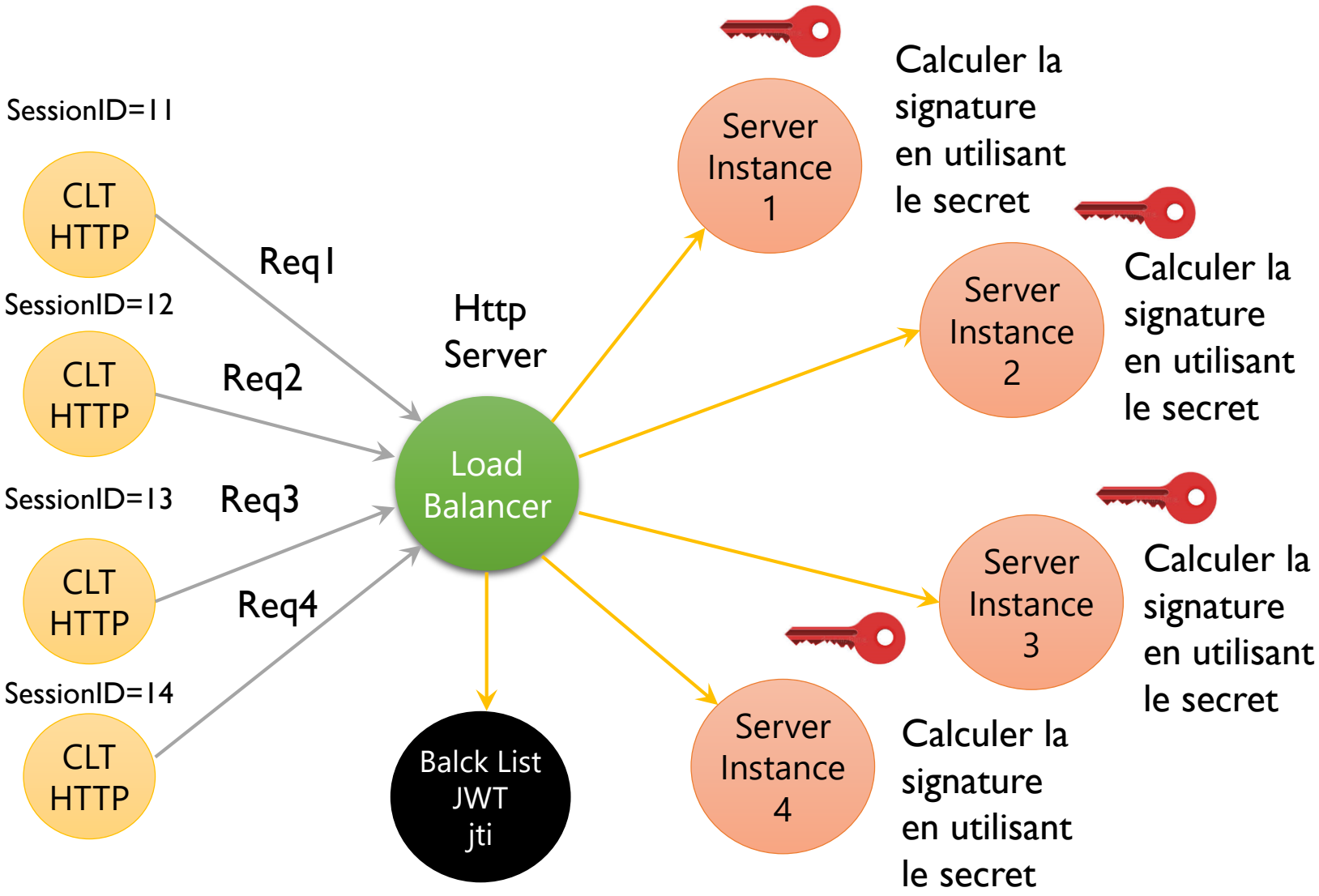
Bonne solution pour les architectures distribuées basées sur les micro services



JWT dans un système distribué avec clé privé et clé publique



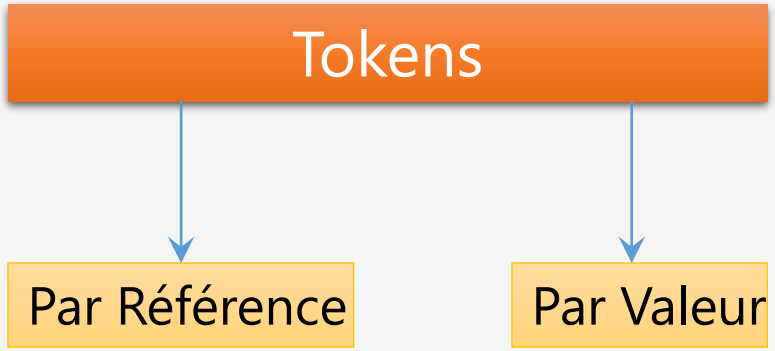
Problème de Révocation des Tokens



Access Token Vs Refresh Token

- Access Token :
  - Contient les informations nécessaires pour accéder directement à une ressource.
  - En d'autres termes, lorsqu'un client transmet un jeton d'accès à un serveur gérant une ressource, ce serveur peut utiliser les informations contenues dans le jeton pour décider si le client est autorisé ou non.
  - Les jetons d'accès ont généralement une date d'expiration et sont de **courte durée**.
- Refresh Token :
  - Contient les informations nécessaires pour obtenir un nouveau Access Token.
  - En d'autres termes, chaque fois qu'un jeton d'accès est requis pour accéder à une ressource spécifique, un client peut utiliser le Refresh Token pour obtenir un nouveau Access Token émis par le serveur d'authentification.
  - Les cas d'utilisation courants incluent
    - L'obtention de nouveaux jetons d'accès après l'expiration des anciens,
    - L'accès à une nouvelle ressource pour la première fois.
  - Les jetons d'actualisation peuvent également expirer mais ont une **durée de vie plutôt longue**.
  - Les jetons d'actualisation sont généralement soumis à des exigences de stockage strictes pour garantir qu'ils ne fuient pas.
  - Ils peuvent également être mis sur liste noire par le serveur d'autorisation.

Session ID Vs JWT



Session ID = 13

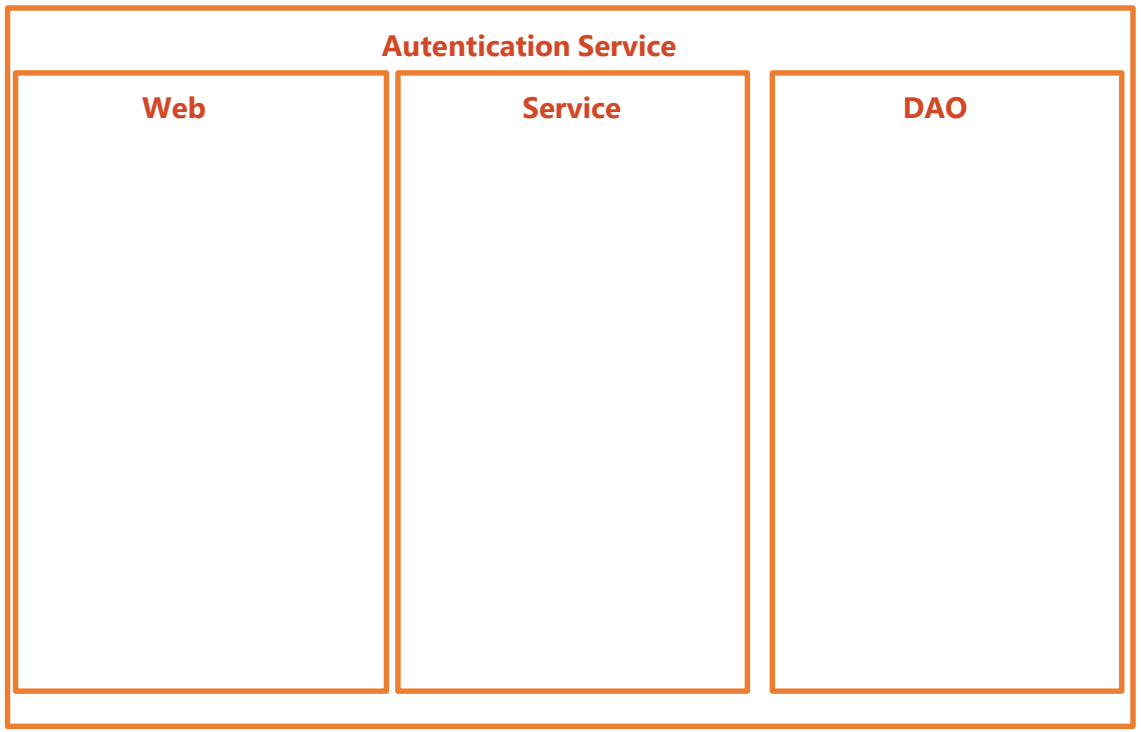
- A besoin d'une application tiers pour connaitre les informations associés à ce Token

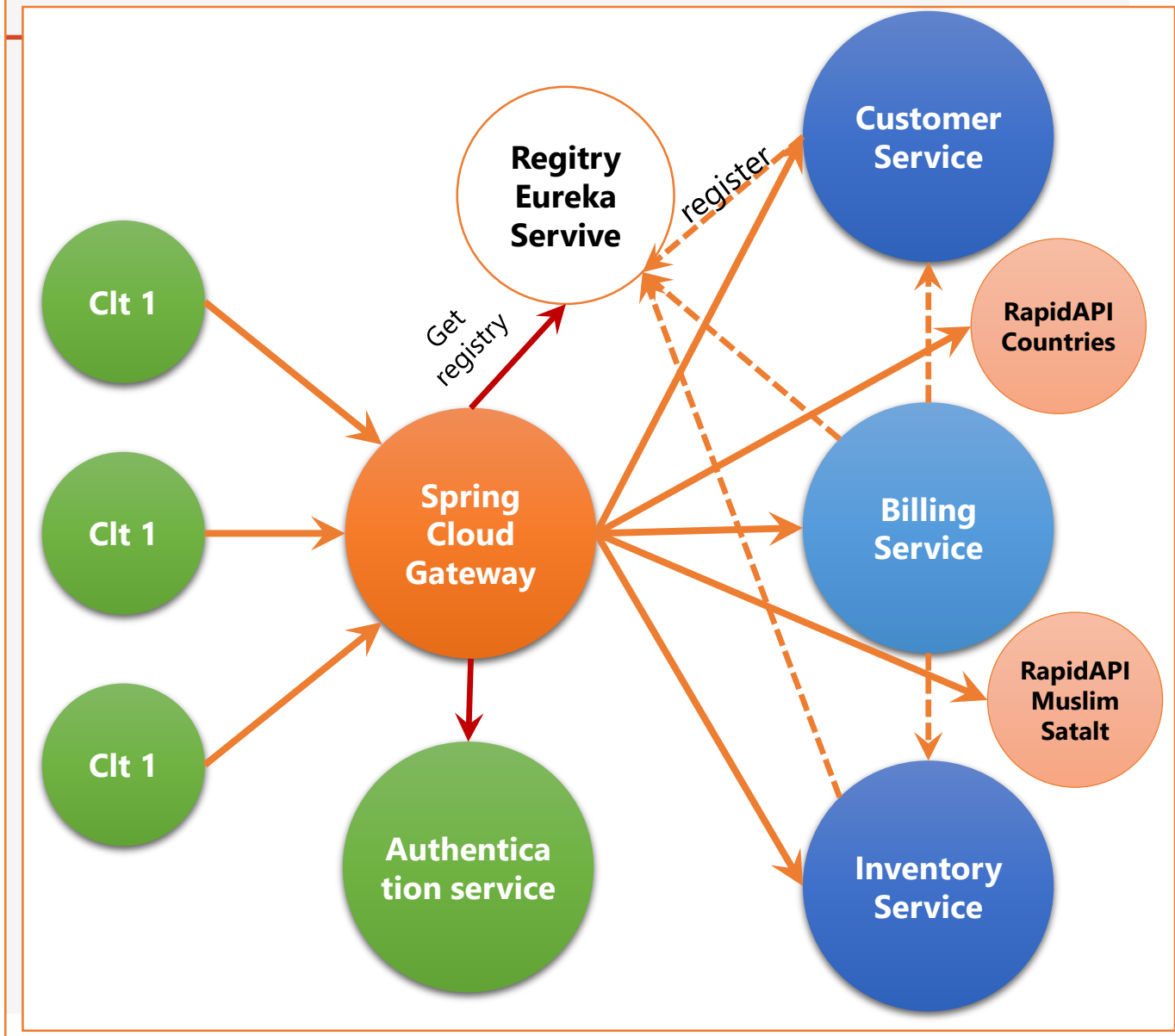
JWT

- N'a pas besoin d'une application tiers pour connaitre les informations associés à ce Token
- Le JWT contient lui-même toutes les informations sur l'utilisateur d'il prétend représenter

Service d'authentification avec Spring Security et JWT

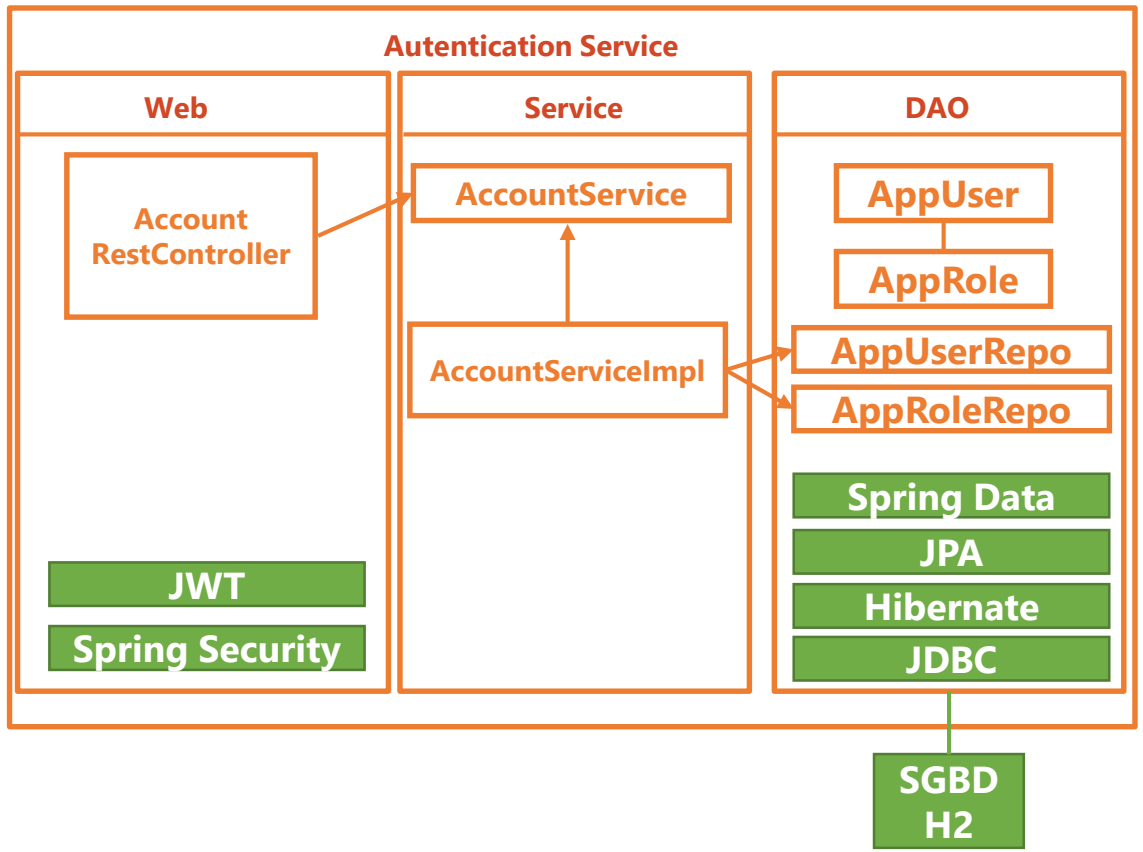
- Créer un micro service d'authentification en utilisant Spring Security et JWT
- Ce services permet de gérer
  - Les utilisateurs
  - Les rôles (USER, ADMIN, CUSTOMER_MANAGER, PRODUCT_MANAGER, BILLS_MANAGER)
  - Un utilisateur peut avoir plusieurs rôles et chaque rôle peut être affecté à plusieurs utilisateurs





TP Sécurité des Micro Services avec Spring Security et JWT

- Créer un micro service d'authentification en utilisant Spring Security et JWT
- Ce services permet de gérer
  - Les utilisateurs
  - Les rôles (USER, ADMIN, CUSTOMER_MANAGER, PRODUCT_MANAGER, BILLS_MANAGER)
  - Un utilisateur peut avoir plusieurs rôles et chaque rôle peut être affecté à plusieurs utilisateurs

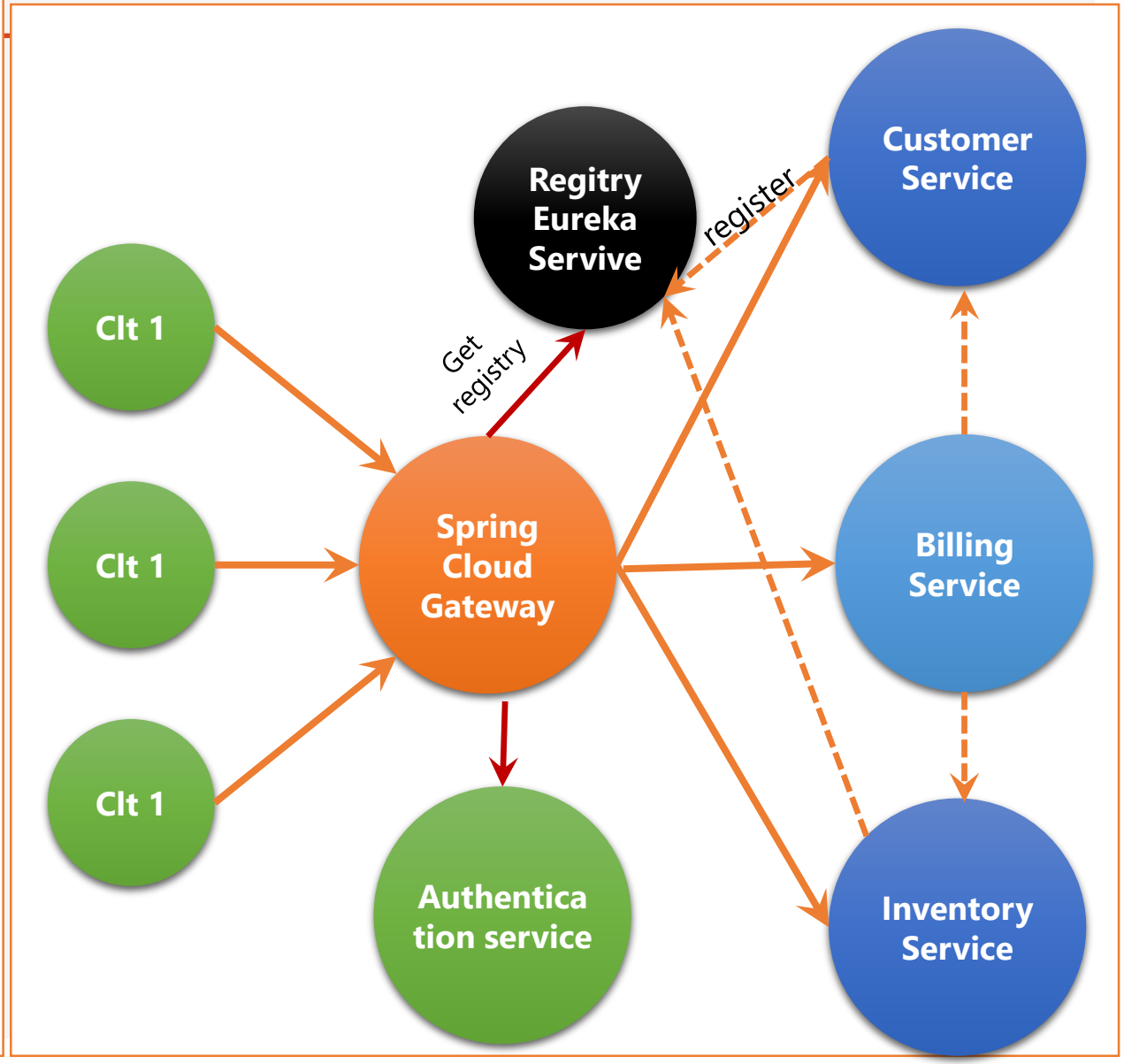


Travail à faire :

1. Créer un Projet Spring Boot avec les dépendances Web, Data JPA, H2, Lombock, Dev Tools, Spring Security
2. Couche DAO
   1. Créer les entités JPA AppUser et AppRole
   2. Créer les interfaces JPA Repository
3. Couche Service
   1. Créer l'interface AccountService déclarant les opérations requises:
      - Ajouter un utilisateur
      - Ajouter un rôle
      - Affecter un rôle à un utilisateur
      - Consulter un utilisateur sachant son username
      - Consulter tous les utilisateurs
      - Consulter tous les rôles
4. Tester les opérations de la couche service
5. Créer un RestController qui permet de gérer les utilisateurs et les rôles
6. Sécuriser le service en instaurant un système d'authentification Stateless avec Spring Security et Json Web Token en délivrant au client le Access Token et le refresh token
7. Tester les opérations du services d'authentification avec un client REST : Advenced Rest Client

Activité Pratique à rendre: Travail à faire

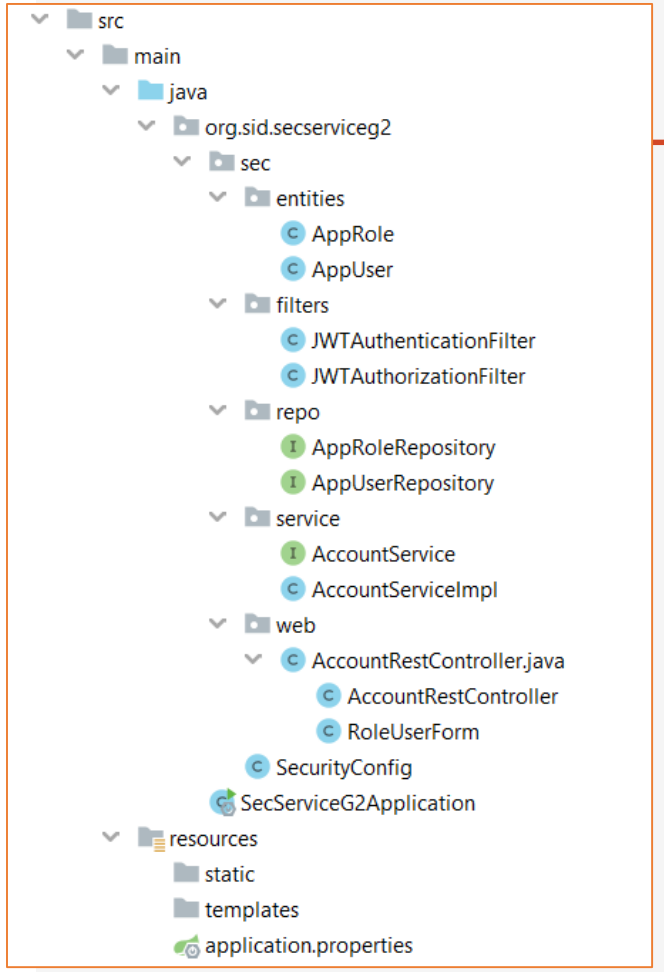
1. Créer le micro service Customer-service
   - Créer l'entité Customer
   - Créer l'interface CustomerRepository basée sur Spring Data
   - Déployer l'API Restful du micro-service en utilisant Spring Data Rest
   - Tester le Micro service
2. Créer le micro service Inventory-service
   - Créer l'entité Product
   - Créer l'interface ProductRepository basée sur Spring Data
   - Déployer l'API Restful du micro-service en utilisant Spring Data Rest
   - Tester le Micro service
3. Créer la Gateway service en utilisant Spring Cloud Gateway
   1. Tester la Service proxy en utilisant une configuration Statique basée sur le fichier application.yml
   2. Tester la Service proxy en utilisant une configuration Statique basée une configuration Java
4. Créer l'annuaire Discovery Service basé sur NetFlix Eureka Server
5. Tester le proxy en utilisant une configuration dynamique de Gestion des routes vers les micro services enregistrés dans l'annuaire Eureka Server
6. Créer le service Billing Service
7. Créer un service d'authentification Statless basé sur Spring Security et Json Web Token. Ce service devrait permettre de :
   - Gérer les utilisateurs et les rôles de l'application
   - Authentifier un utilisateur en lui délivrant un access Token et un refresh Token de type JWT
   - Gérer les autorisation d'accès
   - Renouveler l'access Token à l'aide du refresh Token



Code du Micro Service d'authentification : Dépendances Maven

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.auth0/java-jwt -->
    <dependency>
        <groupId>com.auth0</groupId>
        <artifactId>java-jwt</artifactId>
        <version>3.11.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>
```

Code du Micro Service d'authentification : Couche DAO




```
spring.datasource.url=jdbc:h2:mem:db-users
server.port=8080
```

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class AppRole {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String roleName;
}
```

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class AppUser {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private String password;
    @ManyToMany(fetch = FetchType.EAGER)
    private Collection<AppRole> appRoles=new ArrayList<>();
}
```

```java
public interface AppRoleRepository extends JpaRepository<AppRole,Long> {
    AppRole findByRoleName(String roleName);
}
```

```java
public interface AppUserRepository extends JpaRepository<AppUser,Long> {
    AppUser findByUsername(String username);
}
```

Couche Service

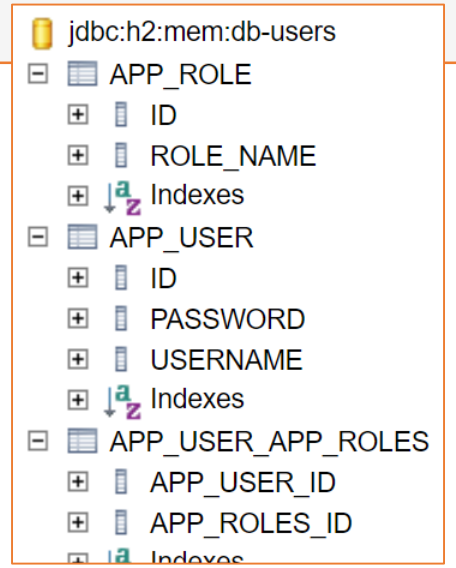
```java
public interface AccountService {
    AppUser addNewUser(AppUser appUser);
    AppRole addNewRole(AppRole appRole);
    void addRoleToUser(String username,String roleName);
    AppUser loadUserByUsername(String username);
    List<AppUser> listUsers();
}
```

```java
@Service
@Transactional
public class AccountServiceImpl implements AccountService {
    private AppUserRepository appUserRepository;
    private AppRoleRepository appRoleRepository;
    private PasswordEncoder passwordEncoder;

    public AccountServiceImpl(AppUserRepository appUserRepository, AppRoleRepository appRoleRepository, PasswordEncoder passwordEncoder)
{
        this.appUserRepository = appUserRepository; this.appRoleRepository = appRoleRepository;
        this.passwordEncoder = passwordEncoder;
    }
    @Override
    public AppUser addNewUser(AppUser appUser) {
        String pw=appUser.getPassword();  appUser.setPassword(passwordEncoder.encode(appUser.getPassword()));
        return appUserRepository.save(appUser);
    }
    @Override
    public AppRole addNewRole(AppRole appRole) {
        return appRoleRepository.save(appRole);
    }
    @Override
    public void addRoleToUser(String username, String roleName) {
        AppUser appUser=appUserRepository.findByUsername(username); AppRole appRole=appRoleRepository.findByRoleName(roleName);
        appUser.getAppRoles().add(appRole);
    }
    @Override
    public AppUser loadUserByUsername(String username) {   return appUserRepository.findByUsername(username) }
    @Override
    public List<AppUser> listUsers() {
        return appUserRepository.findAll();
    }
}
```

Couche Service

```
@SpringBootApplication
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecServiceG2Application {
    public static void main(String[] args) {
        SpringApplication.run(SecServiceG2Application.class, args);
    }
    @Bean
    PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
    @Bean
    CommandLineRunner start(AccountService accountService){
        return args -> {
         accountService.addNewRole(new AppRole(null,"USER"));accountService.addNewRole(new AppRole(null,"ADMIN"));
         accountService.addNewRole(new AppRole(null,"CUSTOMER_MANAGER"));accountService.addNewRole(new AppRole(null,"PRODUCT_MANAGER"));
         accountService.addNewRole(new AppRole(null,"BILLS_MANAGER"));
         accountService.addNewUser(new AppUser(null,"user1","1234",new ArrayList<>()));
         accountService.addNewUser(new AppUser(null,"admin","1234",new ArrayList<>()));
         accountService.addNewUser(new AppUser(null,"user2","1234",new ArrayList<>()));
         accountService.addNewUser(new AppUser(null,"user3","1234",new ArrayList<>()));
         accountService.addNewUser(new AppUser(null,"user4","1234",new ArrayList<>()));
            accountService.addRoleToUser("user1","USER");
            accountService.addRoleToUser("admin","USER");
            accountService.addRoleToUser("user2","USER");
            accountService.addRoleToUser("user3","USER");
            accountService.addRoleToUser("user4","USER");
            accountService.addRoleToUser("admin","ADMIN");
            accountService.addRoleToUser("user2","CUSTOMER_MANAGER");
            accountService.addRoleToUser("user3","PRODUCT_MANAGER");
            accountService.addRoleToUser("user4","BILLS_MANAGER");
        };
    }
}
```



SELECT * FROM APP_USER_APP_ROLES;

APP_USER_ID	APP_ROLES_ID
1	1
2	1
2	2
3	1
3	3
4	1
4	4
5	1
5	5

(9 rows, 4 ms)

SELECT * FROM APP_ROLE;

ID	ROLE_NAME
1	USER
2	ADMIN
3	CUSTOMER_MANAGER
4	PRODUCT_MANAGER
5	BILLS_MANAGER

(5 rows, 4 ms)

SELECT * FROM APP_USER;

ID	PASSWORD	USERNAME
1	$2a$10$D8loCoHL018LFQQr8ysu4ebN7yv7THaciOhW6.j.mdaONriW9AfvS	user1
2	$2a$10$QNgjXvuSU0Xr4OgMamIBLen0TamiZVm0RBzLvOUgGQ8mocHHxLxg.	admin
3	$2a$10$UVBkVqpApoaJUFrn5B60SeTZpjNuqp2LIa4tL7yg3jwiHvR/qW.Nu	user2
4	$2a$10$32EGSrhaSUQvZiZHLnyzw.m4ez9tv5AETtSy1xZesy1LCbmVRsySS	user3
5	$2a$10$ruTb6EYRJ338Z15hKsfMA.sUGPzDk/4FuSZdHCfol/kTr5MWyUIxS	user4

Couche Web : Rest API pour la gestion des utilisateurs et les groupes

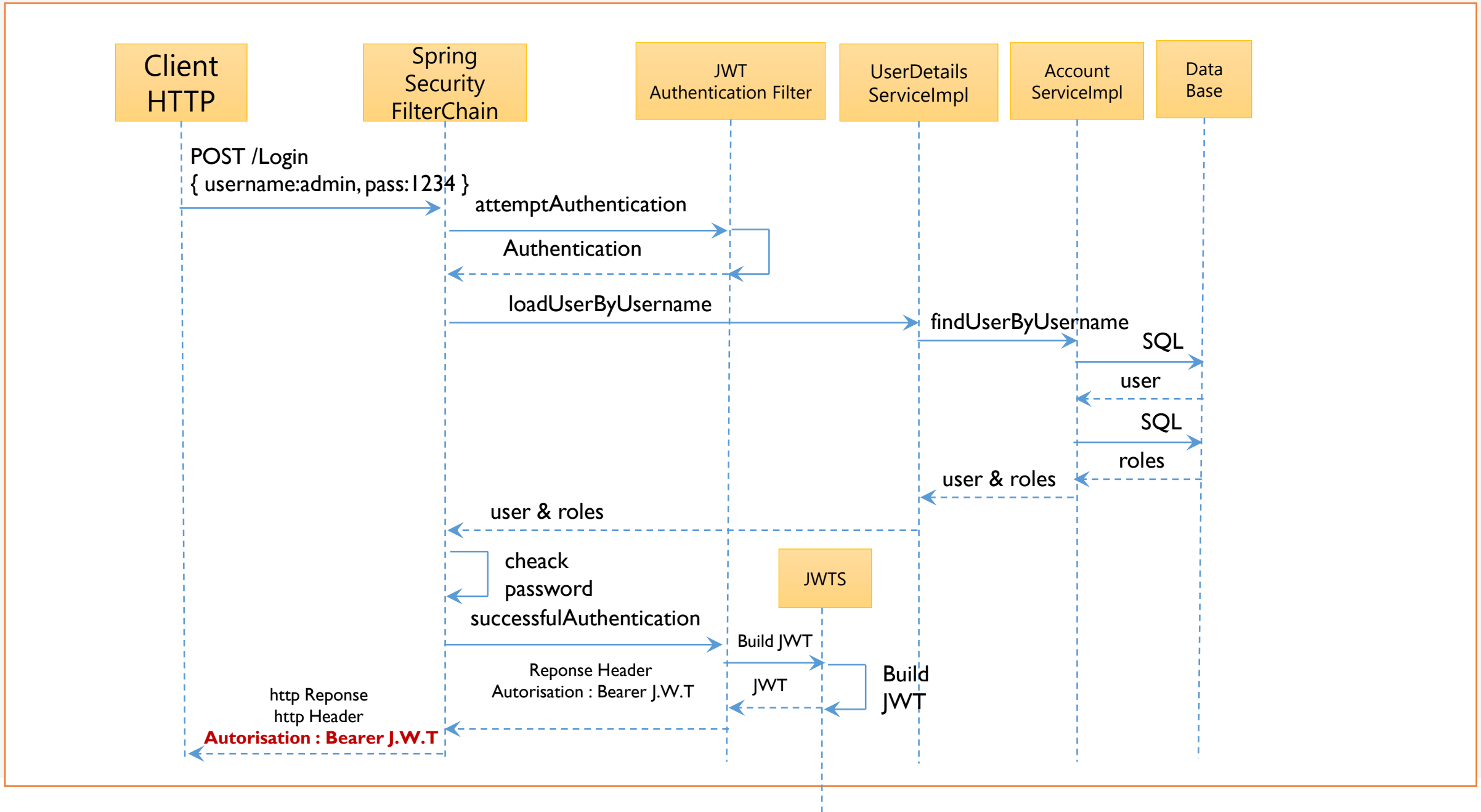
```java
@RestController
public class AccountRestController {
    private AccountService accountService;
    public AccountRestController(AccountService accountService) {
        this.accountService = accountService;
    }
    @GetMapping(path = "/users")
    @PreAuthorize("hasAuthority('USER')")
    public List<AppUser> appUsers() {
        return accountService.listUsers();
    }
    @PostMapping(path = "/users")
    @PreAuthorize("hasAuthority('ADMIN')")
    public AppUser saveUser(@RequestBody AppUser appUser) {
        return accountService.addNewUser(appUser);
    }
    @PostMapping(path = "/roles")
    @PreAuthorize("hasAuthority('ADMIN')")
    public AppRole saveRole(@RequestBody AppRole appRole) {
        return accountService.addNewRole(appRole);
    }
    @PostMapping(path = "/addRoleToUser")
    @PreAuthorize("hasAuthority('ADMIN')")
    public void addRoleToUser(@RequestBody RoleUserForm roleUserForm) {
        accountService.addRoleToUser(roleUserForm.getUsername(), roleUserForm.getRoleName());
    }
```

```java
@Data
class RoleUserForm{
    private String username;
    private String roleName;
}
```

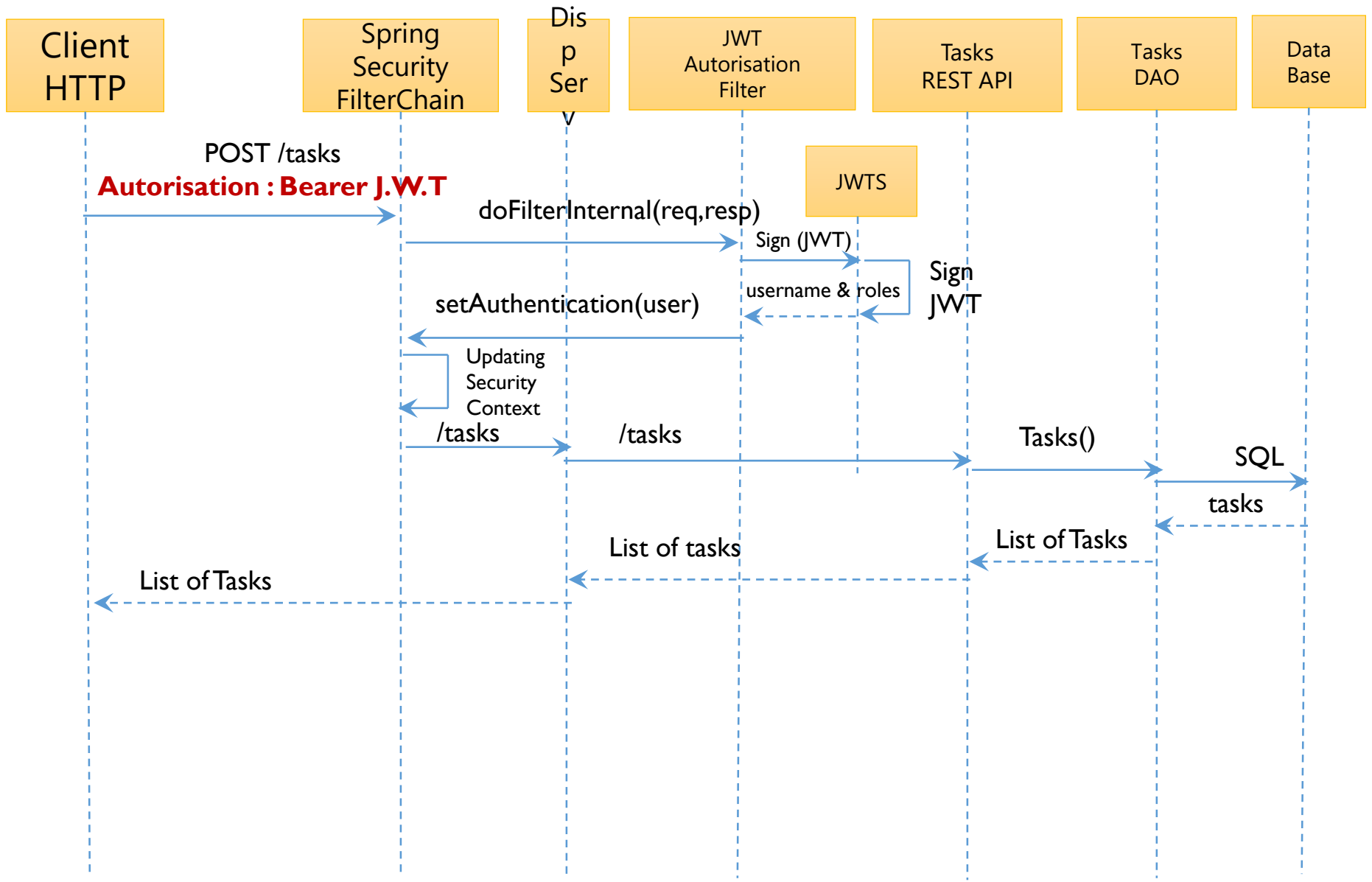
Couche Web : Endpoint de Refresh Token

```java
@PostMapping("/refreshToken")
    public Map<String, String> refreshToken(HttpServletRequest request, HttpServletResponse response) throws IOException {
        String token = request.getHeader("Authorization");
        if (token != null && token.startsWith("Bearer ")) {
            try {
                String jwtRefreshToken = token.substring(7);
                Algorithm algorithm = Algorithm.HMAC256("myHMACPrivateKey");
                JWTVerifier jwtVerifier = JWT.require(algorithm).build();
                DecodedJWT decodedJWT = jwtVerifier.verify(jwtRefreshToken);
                String username = decodedJWT.getSubject();
                AppUser appUser = accountService.loadUserByUsername(username);
                String jwtAccessToken = JWT
                        .create()
                        .withSubject(appUser.getUsername())
                        .withExpiresAt(new Date(System.currentTimeMillis() + 2 * 60 * 1000))
                        .withIssuer(request.getRequestURL().toString())
                        .withClaim("roles", appUser.getAppRoles().stream().map(e -> e.getRoleName()).collect(Collectors.toList()))
                        .sign(algorithm);
                Map<String, String> accessToken = new HashMap<>();
                accessToken.put("Access_Token", jwtAccessToken);
                accessToken.put("Refresh_Token", jwtRefreshToken);
                return accessToken;
            } catch (TokenExpiredException e) {
                response.setHeader("Error-Message", e.getMessage());
                response.sendError(HttpServletResponse.SC_FORBIDDEN);
            }
        }
        throw new RuntimeException("Bad Refresh Token");
    }
}
```

Authentification Spring Security avec JWT



Demander une ressource nécessitant l'authentification



Spring Security Configuration

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private AccountService accountService;
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService((username)->{
            AppUser appUser=accountService.loadUserByUsername(username);
            Collection<GrantedAuthority> authorities= appUser.getAppRoles()
                    .stream().map((role)-> new SimpleGrantedAuthority(role.getRoleName()))
                    .collect(Collectors.toList());
            return new User(appUser.getUsername(),appUser.getPassword(),authorities);
        });
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        //http.formLogin();
        http.authorizeRequests().antMatchers("/login/**","/refreshToken/**","/h2-console/**").permitAll();
        //http.authorizeRequests().antMatchers(HttpMethod.GET,"/users/**").hasAnyAuthority("USER");
        //http.authorizeRequests().antMatchers(HttpMethod.POST,"/users/**").hasAnyAuthority("ADMIN");
        //http.authorizeRequests().antMatchers(HttpMethod.POST,"/roles/**").hasAnyAuthority("ADMIN");
        //http.authorizeRequests().antMatchers(HttpMethod.POST,"/addRoleToUser/**").hasAnyAuthority("ADMIN");
        http.headers().frameOptions().disable();
        http.authorizeRequests().anyRequest().authenticated();
        http.addFilter(new JWTAuthenticationFilter(authenticationManagerBean()));
        http.addFilterBefore(new JWTAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class);
    }
    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception { return super.authenticationManagerBean();} }
```

Spring Security Configuration : JWT Authentication Filter

```
public class JWTAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
    private AuthenticationManager authenticationManager;
    public JWTAuthenticationFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }
    @Override
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws
AuthenticationException {
        AppUser appUser=new AppUser();
        appUser.setUsername(request.getParameter("username"));
        appUser.setPassword(request.getParameter("password"));
        return authenticationManager.authenticate(
          new UsernamePasswordAuthenticationToken(appUser.getUsername(),appUser.getPassword())
        );
    }
```

Spring Security Configuration : JWT Authentication Filter

```
 @Override
    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
Authentication authResult) throws IOException, ServletException {
        User authenticatedUser= (User) authResult.getPrincipal();
        Algorithm algorithm=Algorithm.HMAC256("myHMACPrivateKey");
        String jwtAccessToken= JWT
                .create()
                .withSubject(authenticatedUser.getUsername())
                .withExpiresAt(new Date(System.currentTimeMillis()+2*60*1000))
                .withIssuer(request.getRequestURL().toString())
                .withClaim("roles",authenticatedUser.getAuthorities().stream().map((a)-
>a.getAuthority()).collect(Collectors.toList()))
                .sign(algorithm);
        String jwtRefreshToken= JWT
                .create()
                .withSubject(authenticatedUser.getUsername())
                .withExpiresAt(new Date(System.currentTimeMillis()+10*24*3600*1000))
                .withIssuer(request.getRequestURL().toString())
                .sign(algorithm);
        Map<String,String> accessToken=new HashMap<>();
        accessToken.put("Access_Token",jwtAccessToken);
        accessToken.put("Refresh_Token",jwtRefreshToken);
        response.setContentType("application/json");
        new JsonMapper().writeValue(response.getOutputStream(),accessToken);
    }
}
```

Spring Security Configuration : JWT Authorization Filter

```
public class JWTAuthorizationFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, FilterChain
filterChain) throws ServletException, IOException {
        String token = httpServletRequest.getHeader("Authorization");
        if(token==null || httpServletRequest.getServletPath().equals("/refreshToken")){
            filterChain.doFilter(httpServletRequest,httpServletResponse);
        }else{
            if (token != null && token.startsWith("Bearer ")) {
                try {
                    String jwt = token.substring(7);
                    Algorithm algorithm = Algorithm.HMAC256("myHMACPrivateKey");
                    JWTVerifier jwtVerifier = JWT.require(algorithm).build();
                    DecodedJWT decodedJWT = jwtVerifier.verify(jwt);
                    String username = decodedJWT.getSubject();
                    String[] roles = decodedJWT.getClaim("roles").asArray(String.class);
                    Collection<GrantedAuthority> authorities = new ArrayList<>();
                    for (String role : roles) {
                        authorities.add(new SimpleGrantedAuthority(role));
                    }
                    UsernamePasswordAuthenticationToken authenticationToken =
                            new UsernamePasswordAuthenticationToken(username, null, authorities);
                    SecurityContextHolder.getContext().setAuthentication(authenticationToken);
                    filterChain.doFilter(httpServletRequest, httpServletResponse);
                }
                catch (TokenExpiredException e){
                    httpServletResponse.setHeader("Error-Message",e.getMessage());
                    httpServletResponse.sendError(HttpServletResponse.SC_FORBIDDEN);
                }
            }
            else{
                filterChain.doFilter(httpServletRequest,httpServletResponse);
            }
        }
```

Tests : Authentification

```
curl "http://localhost:8080/login" \
-X POST \  -d "username=admin&password=1234" \
-H "Content-Type: application/x-www-form-urlencoded"
```


```
{
"Refresh_Token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6ODA4MC9sb2dpbiIsImV4cCI6MTYwNzg1NTY0M30.U-uEprAPYJlBju2GMrHCxTOvIcXqV44SCHqE5zW9SdE",

"Access_Token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInJvbGVzIjpbIkFETUlOIiwiVVNFUiJdLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODAvbG9naW4iLCJleHAiOjE2MDY5OTE3NjN9.F7ydrxvyYz50H7V9Ut61sdUoDj1IEi5K4NdmtpYFLlg"
}
```

Tests : Consulter les utilisateurs sans JWT

```
curl "http://localhost:8080/users"
```

```
{
"timestamp":"2020-12-03T10:40:32.298+00:00",
"status":403,
"error":"Forbidden",
"message":"Access Denied",
"path":"/users«
}
```

Tests : Consulter les utilisateurs avec JWT qui a expiré

```
curl "http://localhost:8080/users" \
-H "Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInJvbGVzIjpbIkFETUlOIiwiVVNFUiJdLCJpc3MiOiJodHRwOi8vbG9j
YWxob3N0OjgwODAvbG9naW4iLCJleHAiOjE2MDY5OTE3NjN9.F7ydrxvyYz50H7V9Ut61sdUoDj1IEi5K4NdmtpYFLlg"
```

```
Error-Message: The Token has expired on Thu Dec 03 11:36:03 WEST 2020.
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Content-Length: 0
Date: Thu, 03 Dec 2020 10:43:12 GMT
```

Tests : Consulter les utilisateurs avec un JWT Valide

```
curl "http://localhost:8080/users" \
-H "Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInJvbGVzIjpbIkFETUlOIiwiVVNFUiJdLCJpc3MiOiJodHRwOi8vbG9j
YWxob3N0OjgwODAvbG9naW4iLCJleHAiOjE2MDY5OTI5MDJ9.gVqUNAY6mc_2XlexAUm3QY2lytz_WRpJU_tNGi4_qeY"
```


```
[{"id":1,"username":"user1","appRoles":[{"id":1,"roleName":"USER"}]},
{"id":2,"username":"admin","appRoles":[{"id":1,"roleName":"USER"},{"id":2,"roleName":"ADMIN"}]},
{"id":3,"username":"user2","appRoles":[{"id":1,"roleName":"USER"},{"id":3,"roleName":"CUSTOMER_MANAGER"}]},
{"id":4,"username":"user3","appRoles":[{"id":1,"roleName":"USER"},{"id":4,"roleName":"PRODUCT_MANAGER"}]},
{"id":5,"username":"user4","appRoles":[{"id":1,"roleName":"USER"},{"id":5,"roleName":"BILLS_MANAGER"}]}]
```

Tests : Ajouer un un utilisateur avec le Rôle USER

```
curl "http://localhost:8080/users" \
-X POST \  -d "{\n \"username\":\"azer\",\n  \"password\":\"54321\"\n}" \  -H "Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMSIsInJvbGVzIjpbIlVTRVIiXSwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4
MDgwL2xvZ2luIiwiZXhwIjoxNjA2OTk0MDMyfQ.IYktyPXj0LFJW8mmU_jzus258EE_VmXQQB7ocpiX9no" \
-H "Content-Type: application/json"
```


{
"timestamp": "2020-12-03T11:09:44.191+00:00",
"status": 403,
"error": "Forbidden",
"trace": "org.springframework.security.access.AccessDeniedException: Accès refusé at org.springframework …..
}

Tests : Ajouer un un utilisateur avec le Rôle ADMIN

```
curl "http://localhost:8080/users" \  -X POST \  -d "{\n \"username\":\"azer\",\n  \"password\":\"54321\"\n}" \
-H "Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInJvbGVzIjpbIkFETUlOIiwiVVNFUiJdLCJpc3MiOiJodHRwOi8vbG9j
YWxob3N0OjgwODAvbG9naW4iLCJleHAiOjE2MDY5OTQ1NjR9.eP0kucIbC83ZHUNaNieCfwy0RcuGSxXYxyDGAewzqRA" \  -H "content-
length: 47" \  -H "Content-Type: application/json"
```

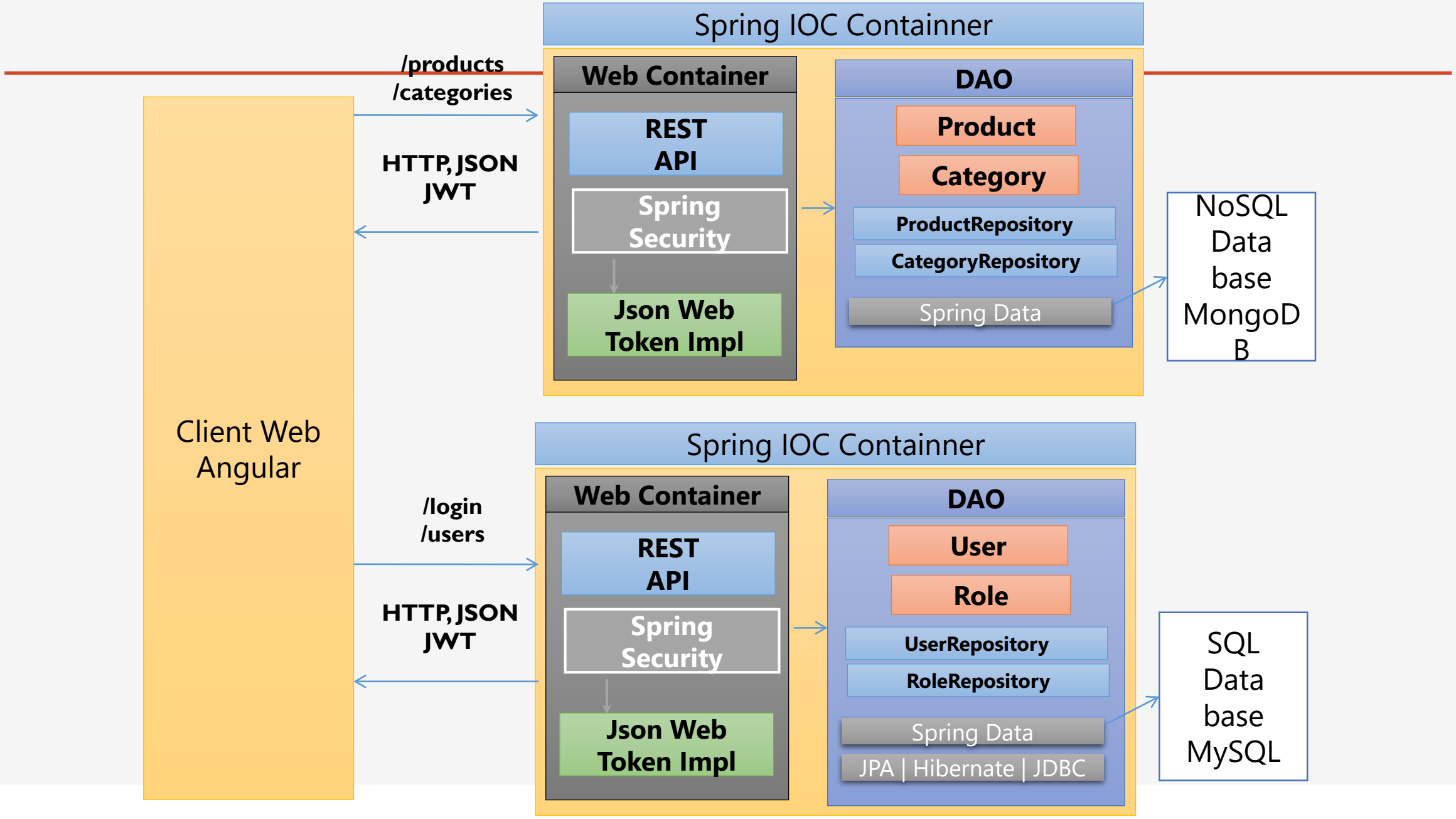

```
{
  "id":7,"username":
  "azer",
  "appRoles":[]
}
```

CORS : JWTAuthorizationFilter

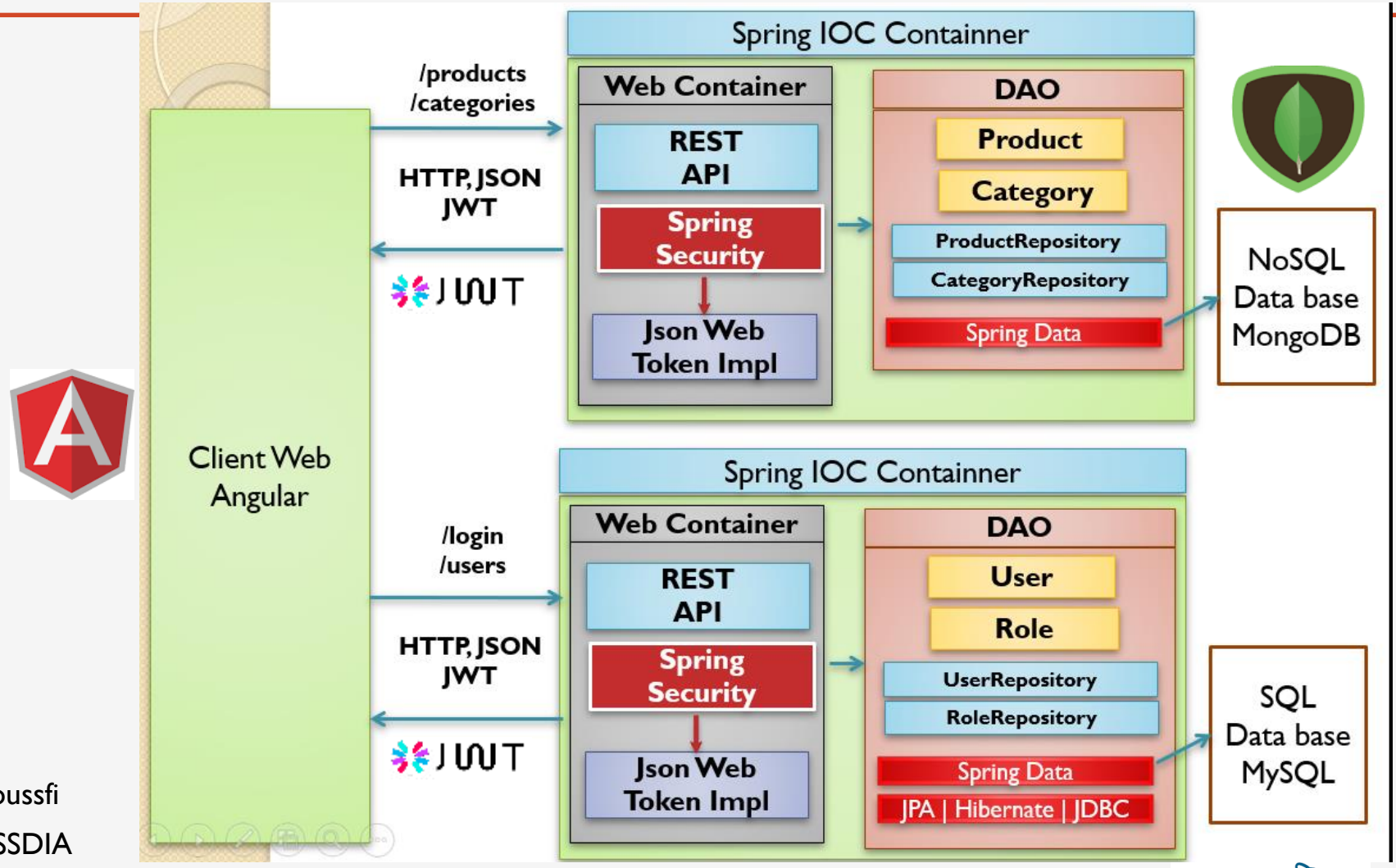
```
public class JWTAuthorizationFilter extends OncePerRequestFilter {


@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
throws IOException, ServletException {
 response.addHeader("Access-Control-Allow-Origin", "*");
 response.addHeader("Access-Control-Allow-Headers", "Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers,authorization");
 response.addHeader("Access-Control-Expose-Headers", "Access-Control-Allow-Origin, Access-Control-Allow-Credentials, authorization");
  if(request.getMethod().equals("OPTIONS")){
    response.setStatus(HttpServletResponse.SC_OK);
  }
else {
  ……..
}}
```

Architecture



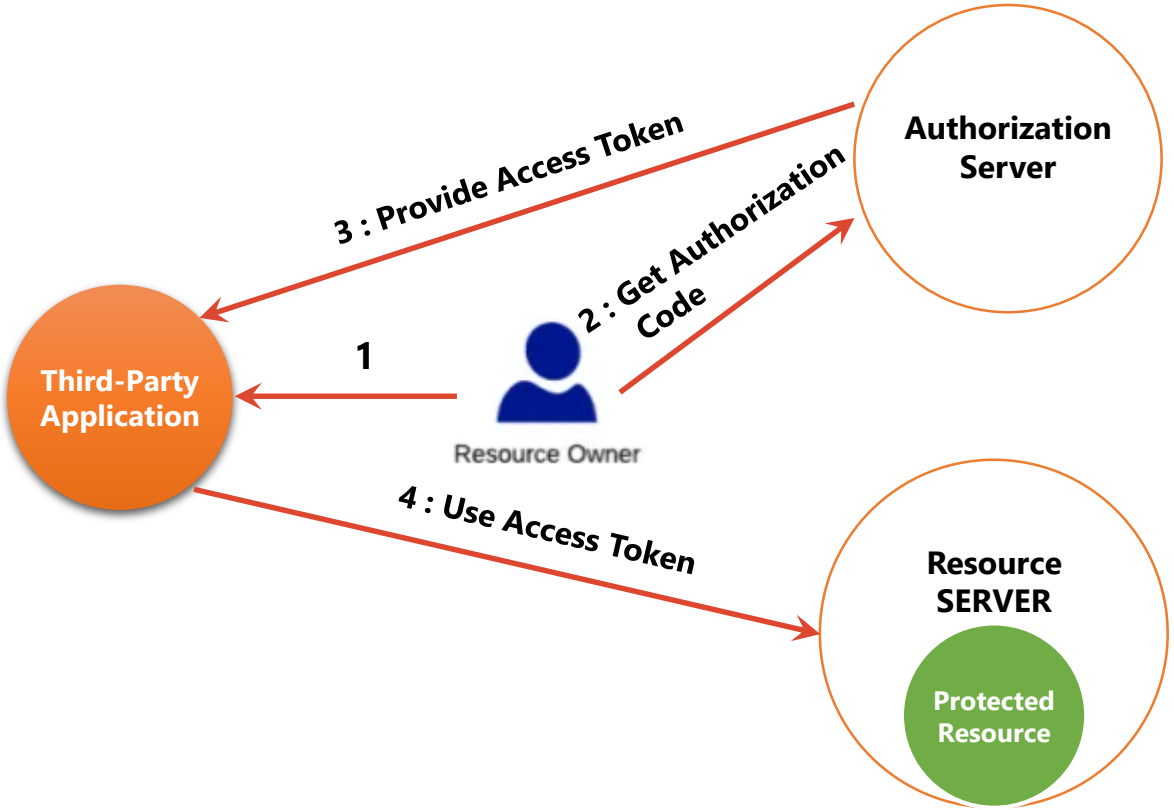
Spring, MongoDB, Security avec JWT, Micro services, Angular



Mohamed Youssfi
Laboratoire SSDIA
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net

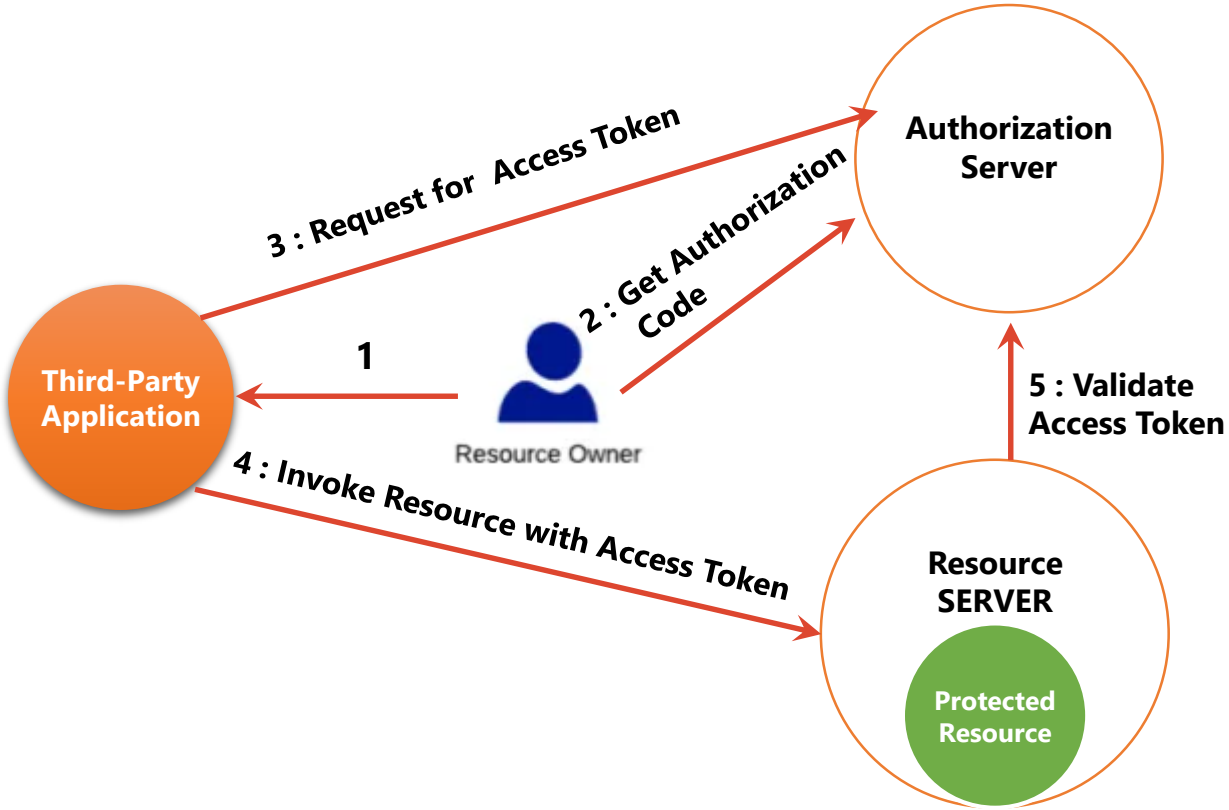
OAuth2

- Protocole de délégation d'autorisations
- Architecture à trois parties
  - Un utilisateur qui possède des ressources exposées par un service en ligne et veut gérer les autorisations d'accès à ces ressources à d'autres applications tierces,
  - Exemple : Autorisation d'une application quelconque X d'accéder aux données de mon compte GitHub, sans obliger l'utilisateur à saisir son mot de passe dans les applications tierce X. Le mot de passe ne doit pas être donné qu'à GitHub.
- Protocole très populaire utilisé par les géants du Web : Facebook, Google, Twiter, etc…
- OAuth2 n'est pas un SSO (Single Sign On et Single Sign Out)
- OAuth2 n'est pas un système de gestion des identités



OpenID Connect 1.0

- Différent de OpenID (1 et 2)
- Développé par OpenID Fundation
- OpenID Connect Core : Novembre 2014
- Construit pour s'adapter aux nouvelles tendances applicatives :
  - Web Server Side
  - Web Client Side
  - Signe Page Application
  - Mobile Applications
  - Systèmes Distribués
  - Web services
  - Micro Services
- Couche au dessus de OAuth2 pour la gestion des identités.
- Introduit le concept de ID Token qui utilise JWT (Json Web Token)



Json Web Token (JWT)

- JSON Web Token (JWT) est un standard (RFC 7519) qui définit une solution **compacte** et **autonome** pour transmettre de manière sécurisée des informations entre les applications en tant qu'objet structuré au format JSON (Java Script Object Notation).
- Cette information peut être vérifiée et fiable car elle est signée numériquement.
- JWT est constitué de trois parties séparées par un point « . » :
  - Header
  - Payload
  - Signature

La forme d'un JWT est donc :

- xxx.yyy.zzz

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgwL2F1dGgiLCJhdWQiOlsiV2ViIEZyb250IEVuZCIsIk1vYmlsZSBBcHAiXSwiZXhwIjo1NDc4OTAwNSwibmJmIjpudWxsLCJpYXQiOjQ5ODY1NDMyLCJqdGkiOiJpZHI1NjU0M2Z0dTg5MDk4NzYiLCJuYW1lIjoibWVkIiwicm9sZXMiOlsiYWRtaW4iLCJhdXRob3IiXX0.-V4FXAPpIBx1HqONU6qp7ptqzq32RrolDlhj4cVUQV0
```

Header

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

Encodage Base 64 URL

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Payload

```
{
  "sub": "1234567890",
  "iss":"http://localhost:8080/auth",
  "aud":["Web Front End","Mobile App"],
  "exp":54789005,
  "nbf":null,
  "iat":49865432,
  "jti":"idr56543ftu8909876",
  "name": "med",
  "roles":["admin","author"]
}
```

Encodage Base 64 URL eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiQmFja2VuZCIsImF1ZCI6WyJXZWIg .... vciJdfQ

Signature : RSA (2 clès publiques et privée) ou HMAC (1 clé privée)

HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

V4FXAPpIBx1HqONU6qp7ptqzq32RrolDlhj4cVUQV0

Oauth 2 Authorization Server

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.0</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>org.sid</groupId>
<artifactId>oauth2-server</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>oauth2-server</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>2020.0.0-M5</spring-cloud.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-security</artifactId>
    </dependency>
```

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

<repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
    </repository>
</repositories>
```

Oauth 2 Authorization Server

```
server.port=8082
#spring.security.user.name=med
#spring.security.user.password=1234
#spring.security.user.roles=USER,ADMIN
#security.oauth2.client.client-id=mobile
#security.oauth2.client.client-secret=pin
#security.oauth2.client.access-token-validity-seconds=3600
#security.oauth2.client.authorized-grant-
types=refresh_token,authorization_code,password,client_credentials
#security.oauth2.client.scope=READ,WRITE
#security.oauth2.authorization.check-token-access=permitAll
```

```
package org.sid.oauth2server;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;

@SpringBootApplication
@EnableAuthorizationServer
public class Oauth2ServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(Oauth2ServerApplication.class, args);
    }
    @Bean
    BCryptPasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

}
```

Oauth 2 Authorization Server

```
package org.sid.oauth2server.config;
import org.springframework.beans.factory.annotation.Autowired;import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.crypto.password.DelegatingPasswordEncoder;import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;
@Configuration
public class OAuth2ServerConfig  extends WebSecurityConfigurerAdapter implements AuthorizationServerConfigurer {
    @Autowired
    BCryptPasswordEncoder passwordEncoder;
    @Override
    public void configure(AuthorizationServerSecurityConfigurer securityConfigurer) throws Exception {
        securityConfigurer.checkTokenAccess("permitAll()");
    }
    @Override
    public void configure(ClientDetailsServiceConfigurer client) throws Exception {
        client
                .inMemory()
                .withClient("mobile").secret(passwordEncoder.encode("1234"))
                .authorizedGrantTypes("password","authorization_code")
                .scopes("READ","WRITE");
    }
    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        endpoints.authenticationManager(authenticationManagerBean());
    }
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

Oauth 2 Authorization Server

```
package org.sid.oauth2server.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.authentication.configuration.GlobalAuthenticationConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class UserConfiguration extends GlobalAuthenticationConfigurerAdapter {
    @Autowired
    PasswordEncoder passwordEncoder;
    @Override
    public void init(AuthenticationManagerBuilder auth) throws Exception {
      auth.inMemoryAuthentication()
              .withUser("admin").password(passwordEncoder.encode("1234"))
              .roles("USER","ADMIN","MANAGER")
               .authorities("CAN_READ","CAN_WRITE","CAN_DELETE","CAN_UPDATE").and()
            .withUser("user").password(passwordEncoder.encode("1234"))
              .roles("USER")
              .authorities("CAN_READ");
    }
}
```

Authentification et validation du token

Authentification

```
curl -d "grant_type=password&username=user&password=1234" -X POST http://localhost:8082/oauth/token -H "Content-Type: application/x-www-form-urlencoded" -u mobile:1234
```

```
{
 "access_token":"6ab5a186-294b-4046-8242-1af0134540db",
 "token_type":"bearer",
 "expires_in":40903,
 "scope":"READ WRITE"
}
```

Validation du token

curl http://localhost:8082/oauth/check_token?token=6ab5a186-294b-4046-8242-1af0134540db

{"active":true,"exp":1606023818,"user_name":"user","authorities":["CAN_READ"],"client_id":"mobile","scope":["READ","WRITE"]}

curl http://localhost:8082/oauth/check_token?token=6ab5a186-294b-4046-8242-1af0134540db | json_pp -json_opt pretty,canonical

```
{
   "active" : true,
   "authorities" : [
      "CAN_READ"
   ],
   "client_id" : "mobile",
   "exp" : 1606023818,
   "scope" : [
      "READ",
      "WRITE"
   ],
   "user_name" : "user"
}
```

Oauth 2 Resource Server

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-security</artifactId>
    </dependency>
```

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Oauth 2 Resource Server

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
class Compte{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY )
    private Long id;
    private String type;
    private double solde;
}
```

```java
@RepositoryRestResource
interface CompteRepository extends JpaRepository<Compte,Long> {
}
```

```java
@SpringBootApplication
@EnableResourceServer
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class DemoComptesApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoComptesApplication.class, args);
    }
@Bean
    CommandLineRunner start(CompteRepository compteRepository){
        String[] types={"courant","epargne"};
        return args -> {
            for (int i = 0; i <10 ; i++) {
                String type=types[new Random().nextInt(1)];
                compteRepository.save(new
Compte(null,type,Math.random()*9000));
            }
        };
    }
}
```

```
server.port=8081
spring.datasource.url=jdbc:h2:mem:comptes-db
security.oauth2.resource.token-info-uri=http://localhost:8082/oauth/check_token
security.oauth2.client.client-id=mobile
security.oauth2.client.client-secret=1234
```

Oauth 2 Resource Server

```java
@RestController
@RequestMapping("/api")
class BanqueRestController{
    @Autowired
    private CompteRepository compteRepository;
    @GetMapping(path = "/comptes/{id}")
    @PreAuthorize("hasAuthority('CAN_READ')")
    public Compte getCompte(@PathVariable Long id){
        return compteRepository.findById(id).get();
    }
    @PostMapping(path = "/comptes")
    @PreAuthorize("hasAuthority('CAN_WRITE')")
    public Compte save(@RequestBody Compte cp){
        return compteRepository.save(cp);
    }
    @PutMapping(path = "/comptes/{id}")
    @PreAuthorize("hasAuthority('CAN_UPDATE')")
    public Compte update(@PathVariable Long id,@RequestBody Compte cp){
        cp.setId(id);
        return compteRepository.save(cp);
    }
    @DeleteMapping(path = "/comptes/{id}")
    @PreAuthorize("hasAuthority('CAN_DELETE')")
    public void delete(@PathVariable Long id){
        compteRepository.deleteById(id);
    }
}
```

Oauth 2 Resource Server

```
package org.sid.democomptes;

import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;

@Configuration
public class HasAutorityConfig extends ResourceServerConfigurerAdapter {
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers(HttpMethod.POST,"/comptes/**").hasAuthority("CAN_WRITE")
                .anyRequest().authenticated();
    }
}
```

Oauth 2 Resource Server

```
package org.sid.democomptes;

import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;

@Configuration
public class HasAutorityConfig extends ResourceServerConfigurerAdapter {
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers(HttpMethod.POST,"/comptes/**").hasAuthority("CAN_WRITE")
                .anyRequest().authenticated();
    }
}
```

Accès à la ressource

```
curl -d "grant_type=password&username=user&password=1234" -X POST
http://localhost:8082/oauth/token -H "Content-Type: application/x-www-form-urlencoded" -
u mobile:1234
```


```
{
 "access_token":"6ab5a186-294b-4046-8242-1af0134540db",
 "token_type":"bearer",
 "expires_in":40903,
 "scope":"READ WRITE"
}
```


```
curl http://localhost:8081/api/comptes/1 -H "Content-Type: application/x-www-
form-urlencoded" -H "Authorization:bearer 6ab5a186-294b-4046-8242-
1af0134540db"
```


```
{"id":1,"type":"courant","solde":6016.382751181522}
```