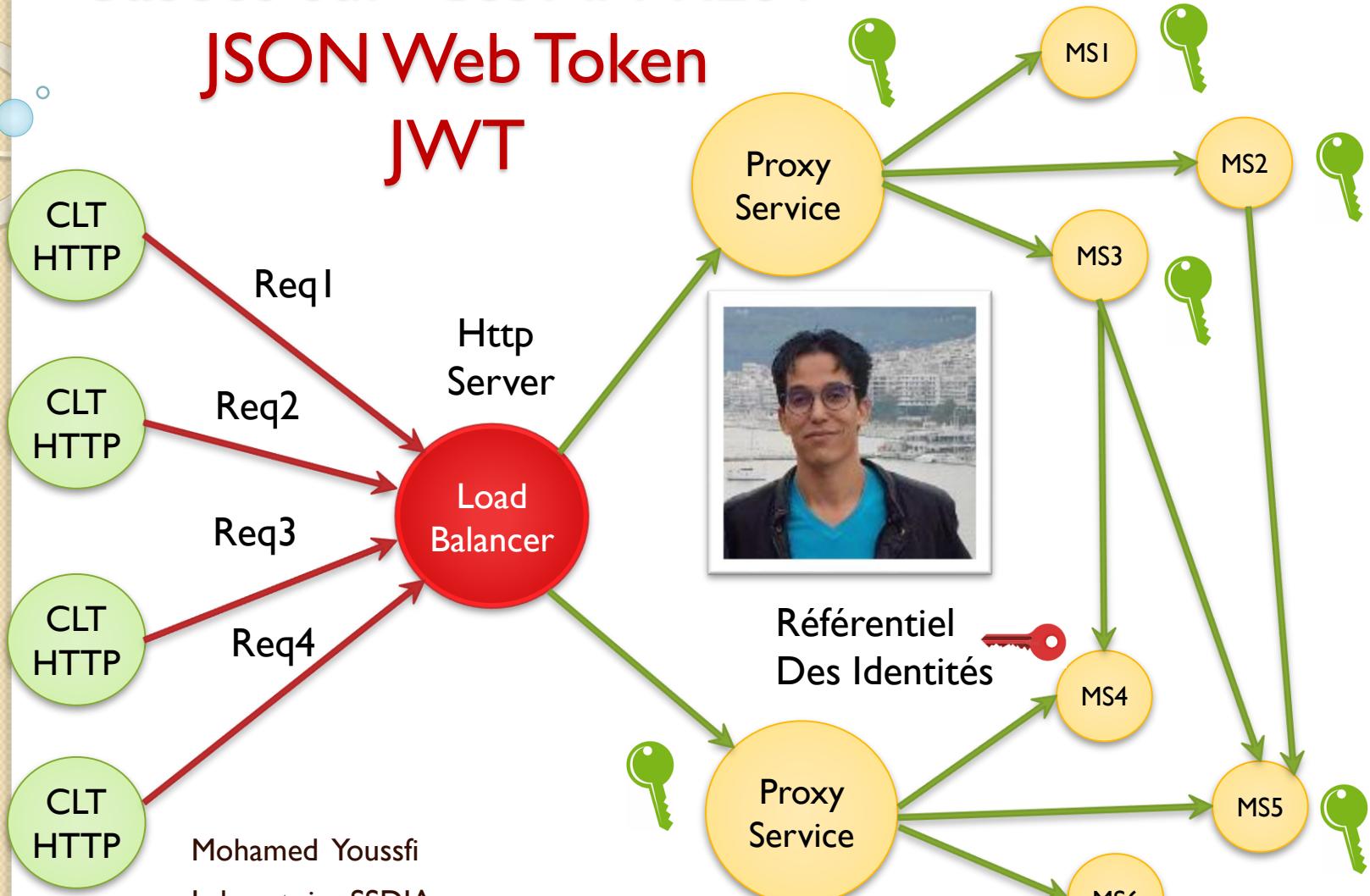


Sécurité des applications web

basées sur des API REST

JSON Web Token JWT



Mohamed Youssfi

Laboratoire SSDIA

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

Sécurité des applications web basée sur des API REST

JSON Web Token JWT

Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

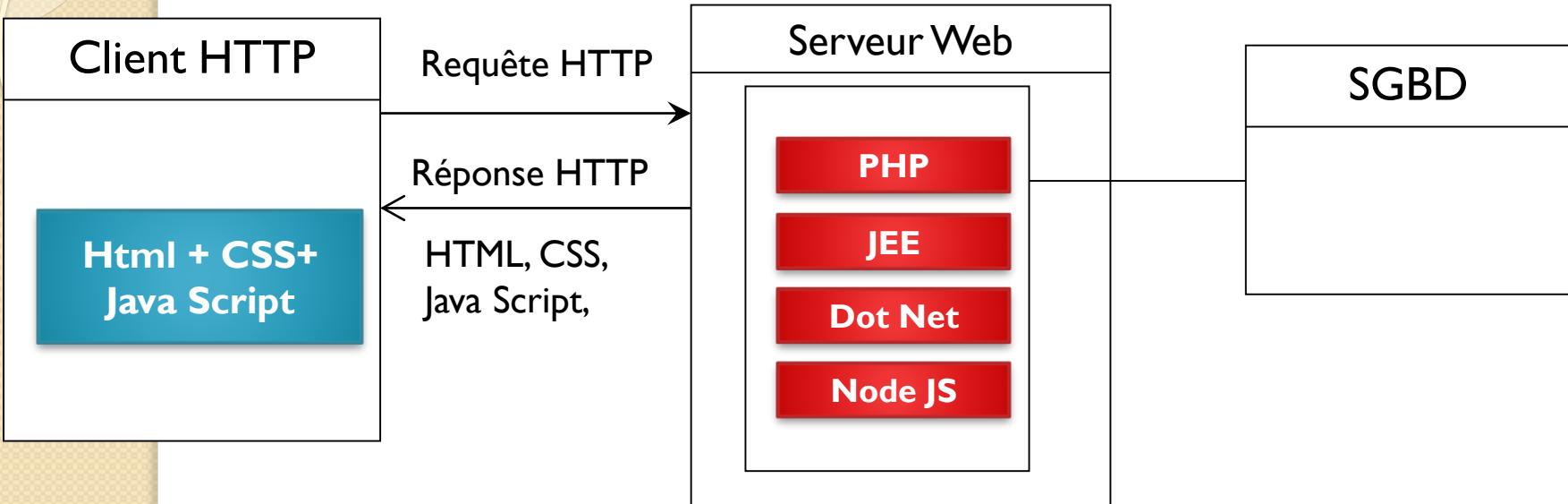
Email : med@youssfi.net

Supports de cours : <http://fr.slideshare.net/mohamedyoussfi9>

Chaîne vidéo : <http://youtube.com/mohamedYoussfi>

Recherche : http://www.researchgate.net/profile/Youssi_Mohamed/publications

Architecture Web



- Un client web (Browser) communique avec le serveur web (Apache) en utilisant le protocole HTTP
- Une application web se compose de deux parties:
 - La partie Backend : S'occupe des traitements effectués coté serveur :
 - Technologies utilisées : PHP, JEE, .Net, Node JS
 - La partie Front end : S'occupe de la présentations des IHM coté Client :
 - Langages utilisés : HTML, CSS, Java Script
- La communication entre la partie Frontend et la partie Backend se fait en utilisant le protocole HTTP

Systèmes d'authentification

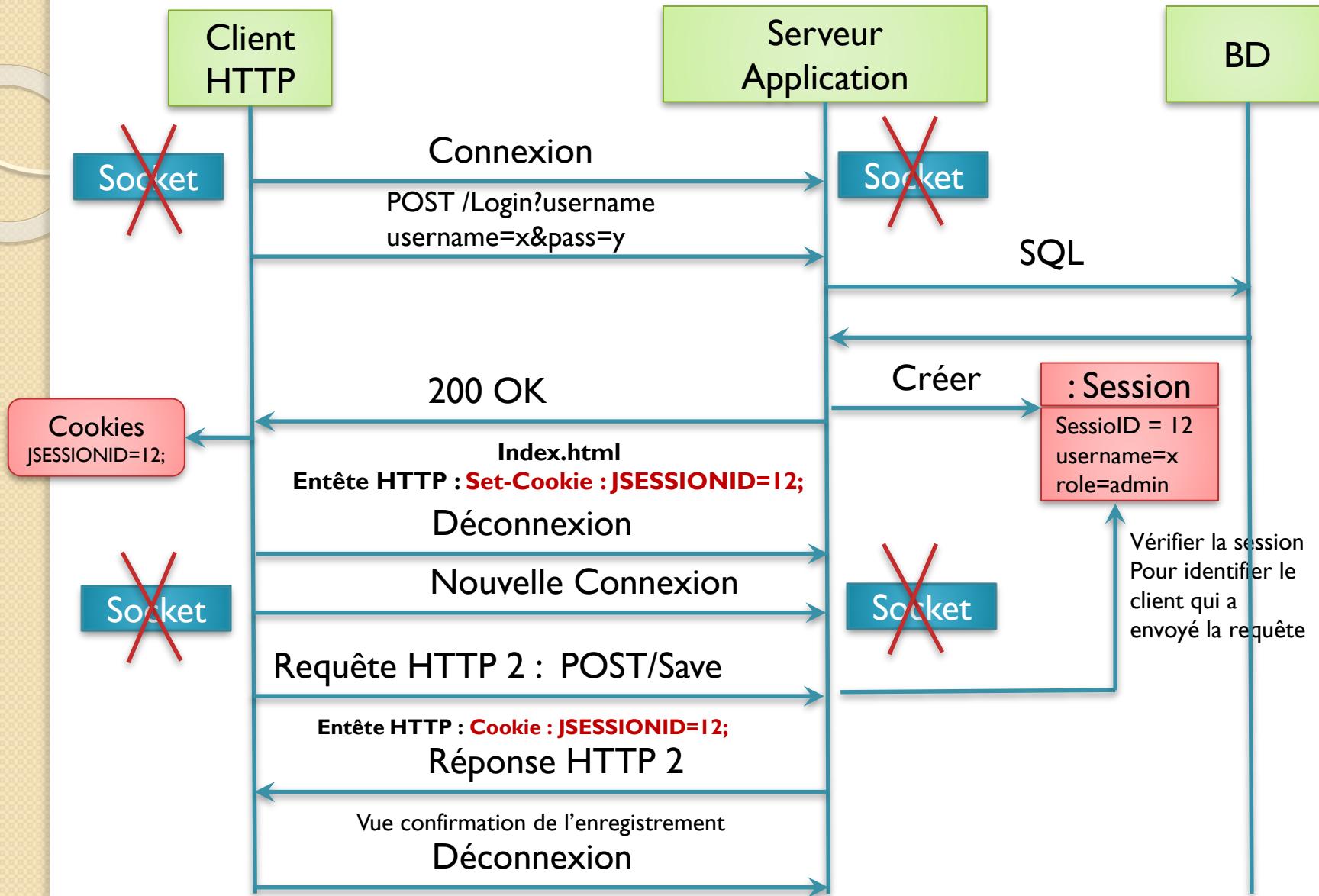
- Deux types :
 - Statful : Les données de la session sont enregistrés côté serveur d'authentification
 - Stateless : les données de la session sont enregistrés dans un jeton d'authentification délivré au client.



Authentification Statful basée sur

Les sessions et Cookies

Sécurité basée sur 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 connaît 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 où Yassine reçoit 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 côté 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 présentations :
 - 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.

Type d'attaque: Cross Site Scripting (XSS)

- Le **Cross-site Scripting** (abrégé **XSS**) est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page.
- Les attaquants exploitent les sites vulnérables à ce type d'attaque pour :
 - Saisir dans les formulaires de ce site, des données contenant des scripts (Par exemple Java Script). Ces scripts sont insérés dans le contenu du site
 - Une fois ce contenu affiché par les navigateurs de ce site, ces scripts sont exécutés dans la machine du navigateur provoquant des actions dramatiques pour l'utilisateur de ce site :
 - modifier ou d'ajouter des clefs à la base de registre de la machine victime ;
 - Afficher une fenêtre demandant à l'utilisateur de rentrer son login et son mot de passe puis de valider, après quoi le résultat sera envoyé par courriel à l'attaquant ;
 - Voler les cookies non sécurisés présents sur la machine victime ;
 - Exécuter des commandes systèmes ;
 - Rediriger vers un autre site pour de l'hameçonnage (**phishing**)
- Le cross-site scripting est abrégé XSS pour ne pas être confondu avec le CSS (feuilles de style), X se lisant « cross » (croix) en anglais.

Parades de XSS

• RECOMMANDATION SUR LES SERVEURS

- La vulnérabilité est fondamentalement l’acceptation sans vérification de données fournies par l’internaute et leur renvoi tel quels.
- Pour contrer les attaques basées sur l’injection de code indirecte, il convient donc de filtrer ces données :
 - vérification syntaxique des entrées, avec suppression des caractères ne pouvant intervenir dans des entrées légitimes. (<), (>), (/), (Script) etc. ;
 - Utiliser les cookies en mode **HttpOnly** pour ne pas permettre aux scripts de lire ces cookies contenant par exemple les SessionID.
 - Transcodage des caractères (signe inférieur transformé en <, par exemple) pour un renvoi sans interprétation à l’internaute ;
 - La surveillance des journaux de connexion et des journaux d’interrogation des bases de données (sites dynamiques) peut indiquer des tentatives et des réussites d’exploitation de ces vulnérabilités.
 - Personnaliser les messages de exceptions générées par le site pour ne pas donner d’indication à exploiter par l’attaquant.

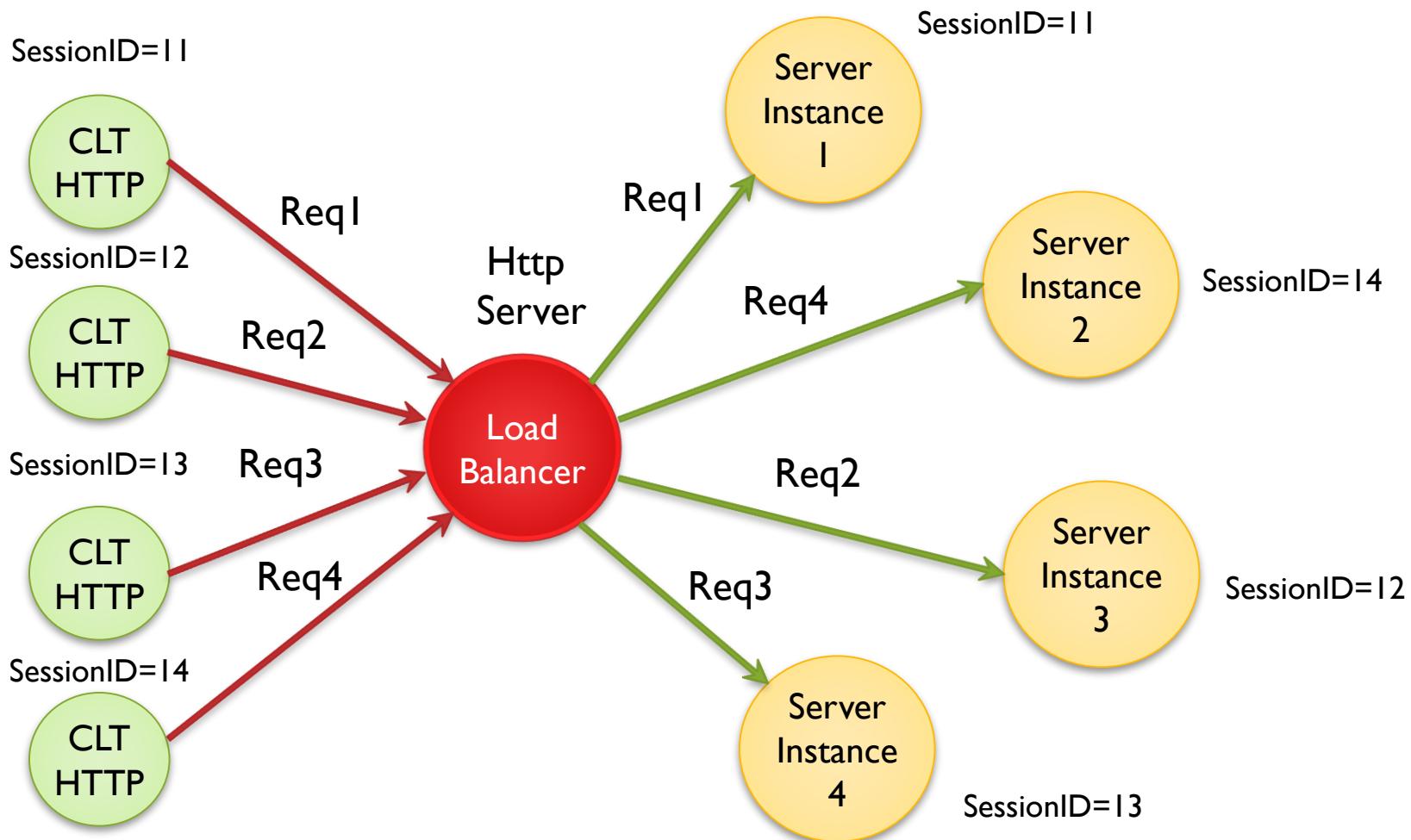
Parades de XSS

- RECOMMANDATION SUR LES NAVIGATEURS
 - L'utilisateur ne peut corriger cette vulnérabilité du serveur, mais en restreindre la possibilité d'exploitation.
 - Il est recommandé à l'internaute :
 - Regarder bien le nom de domaine de son URL
 - Ne pas cliquer sur des liens hypertexte provenant de sources non sûres ;
 - Utiliser son ordinateur (navigation, messagerie, etc.) avec des droits limités ;
 - Visualiser ses courriels en texte brut, de manière à voir les liens réels cachés derrière des textes ou des liens apparents.

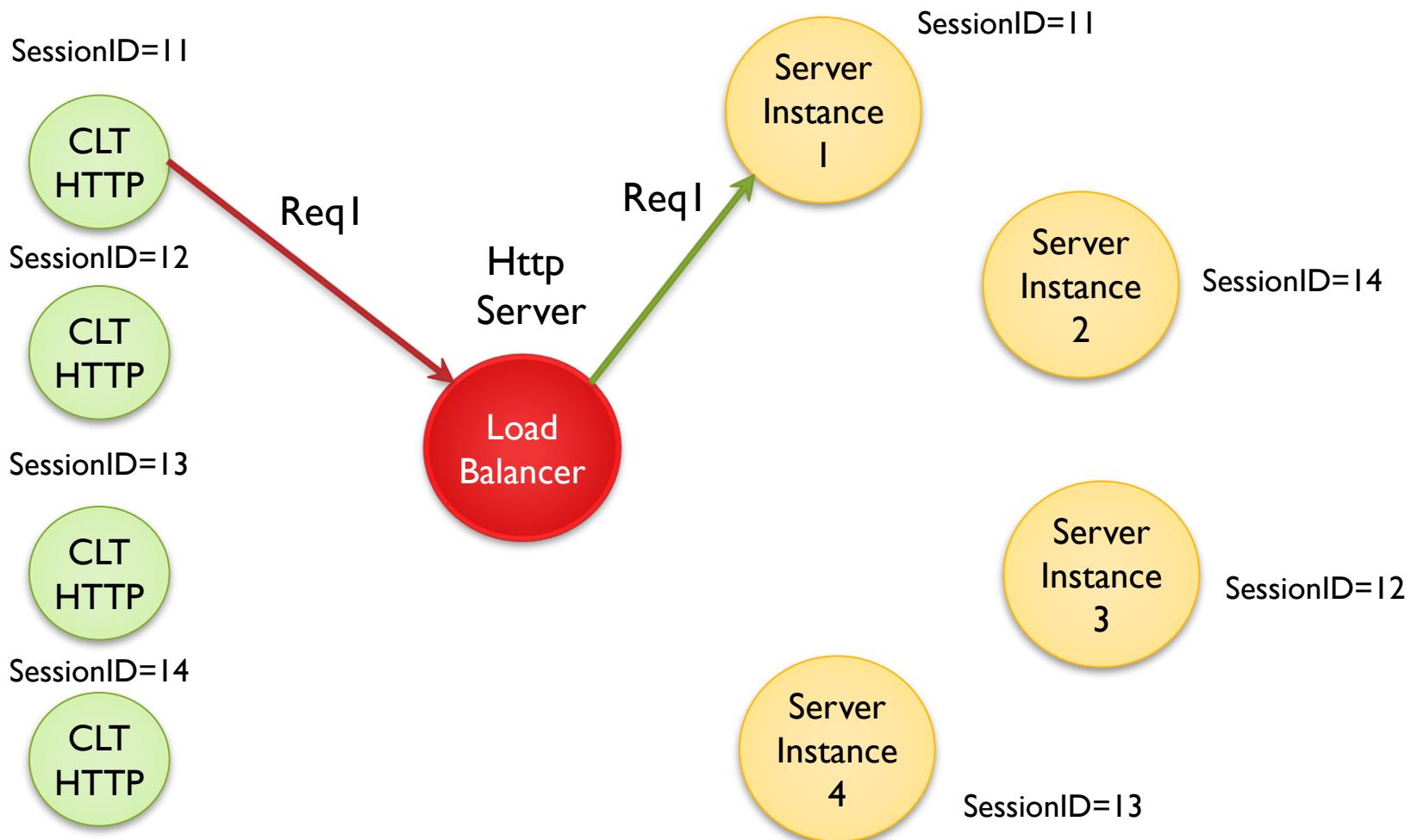
Authentification basée sur les Sessions :

Problème de montée en charge

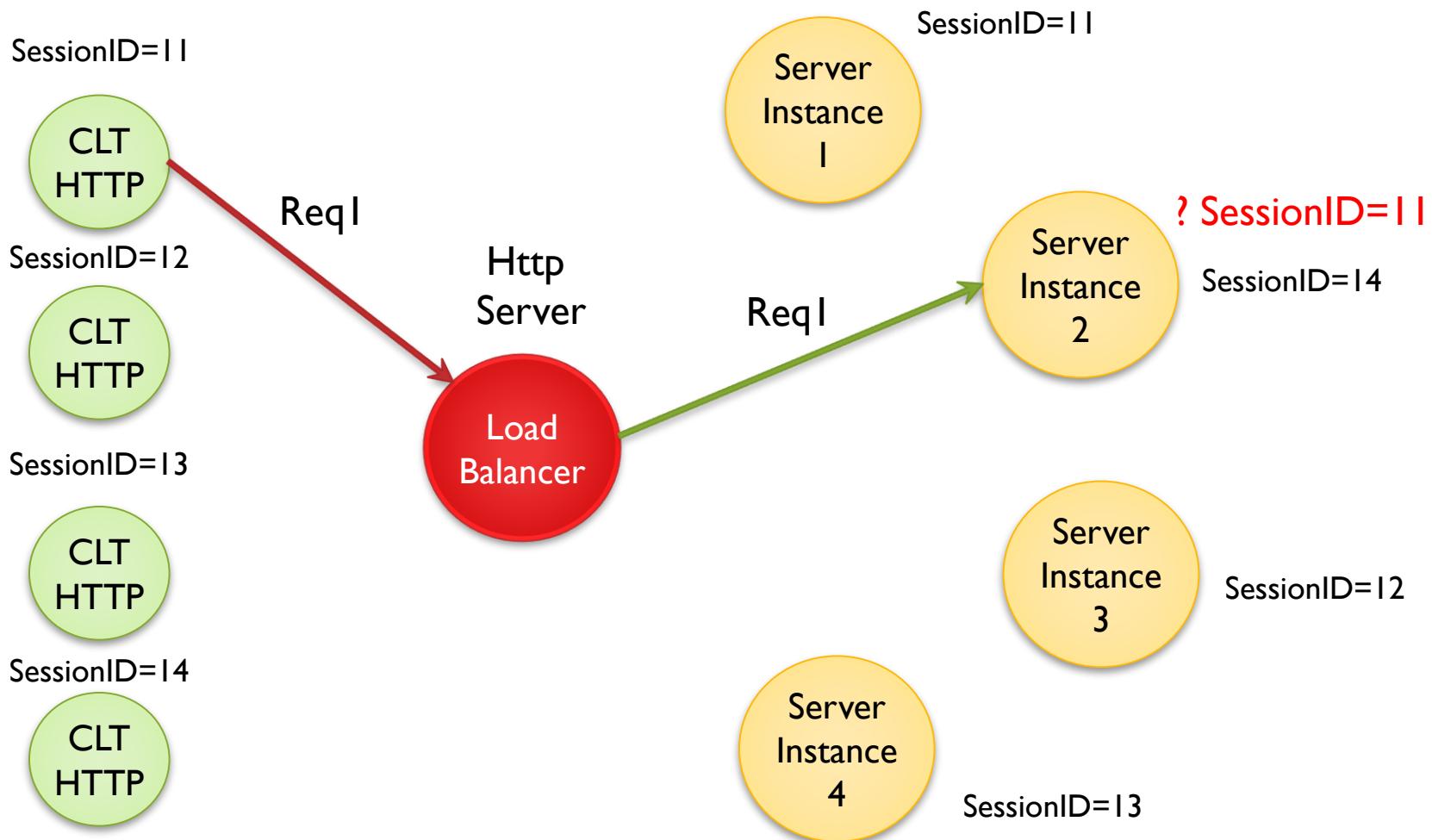
Sécurité basée sur les sessions : Problème de montée en charge



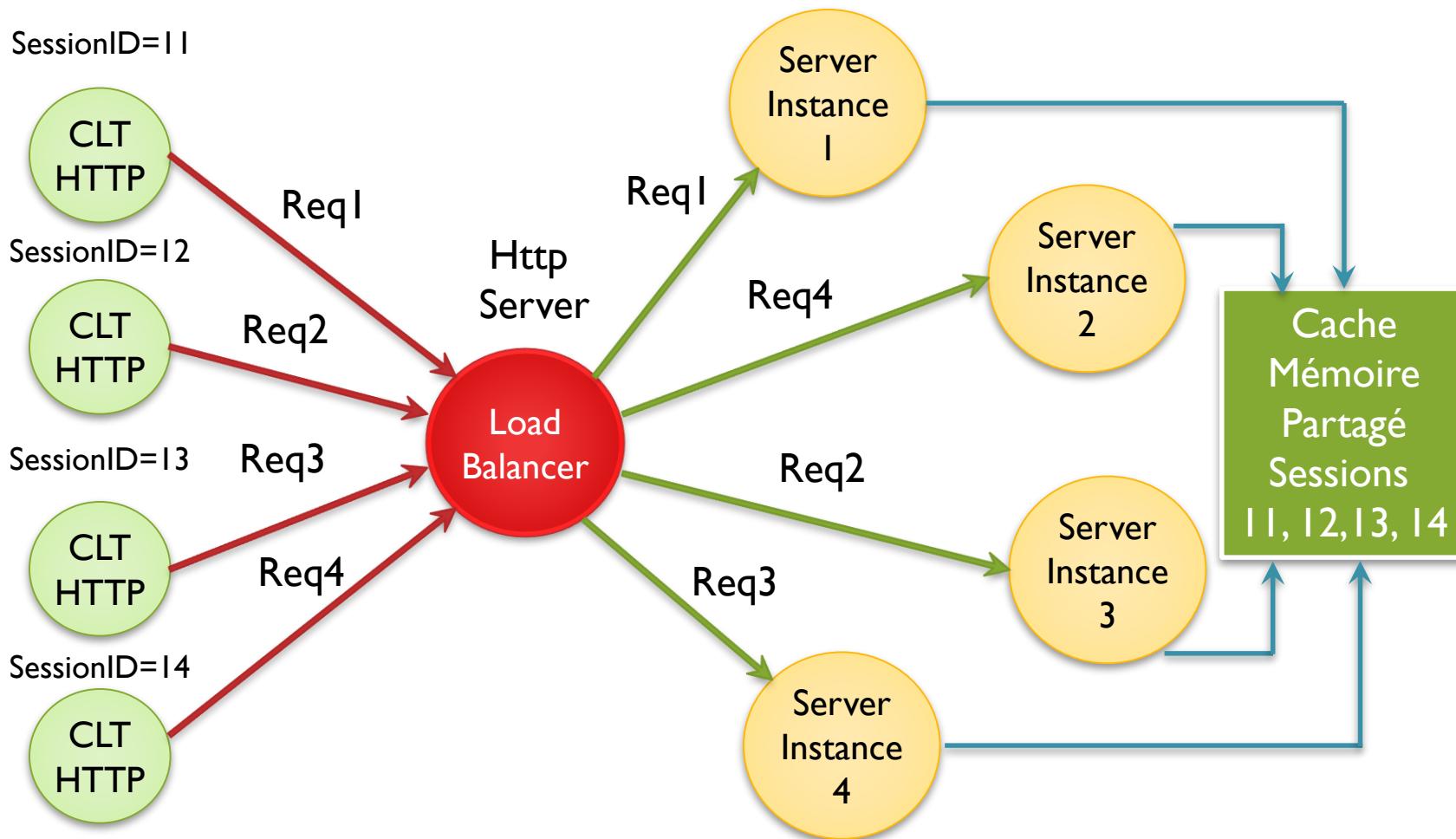
Sécurité basée sur les sessions : Problème de montée en charge



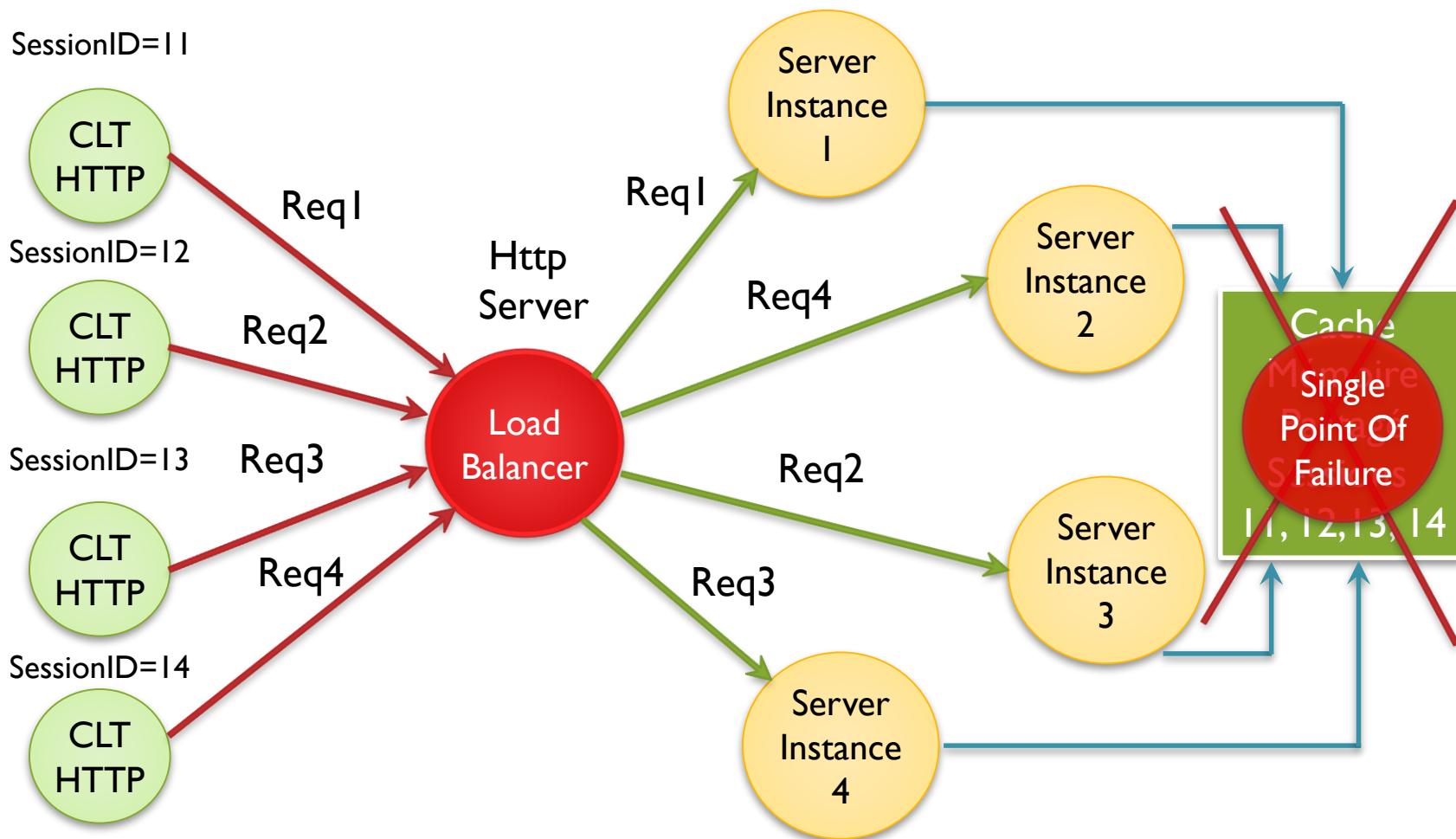
Sécurité basée sur les sessions : Problème de montée en charge



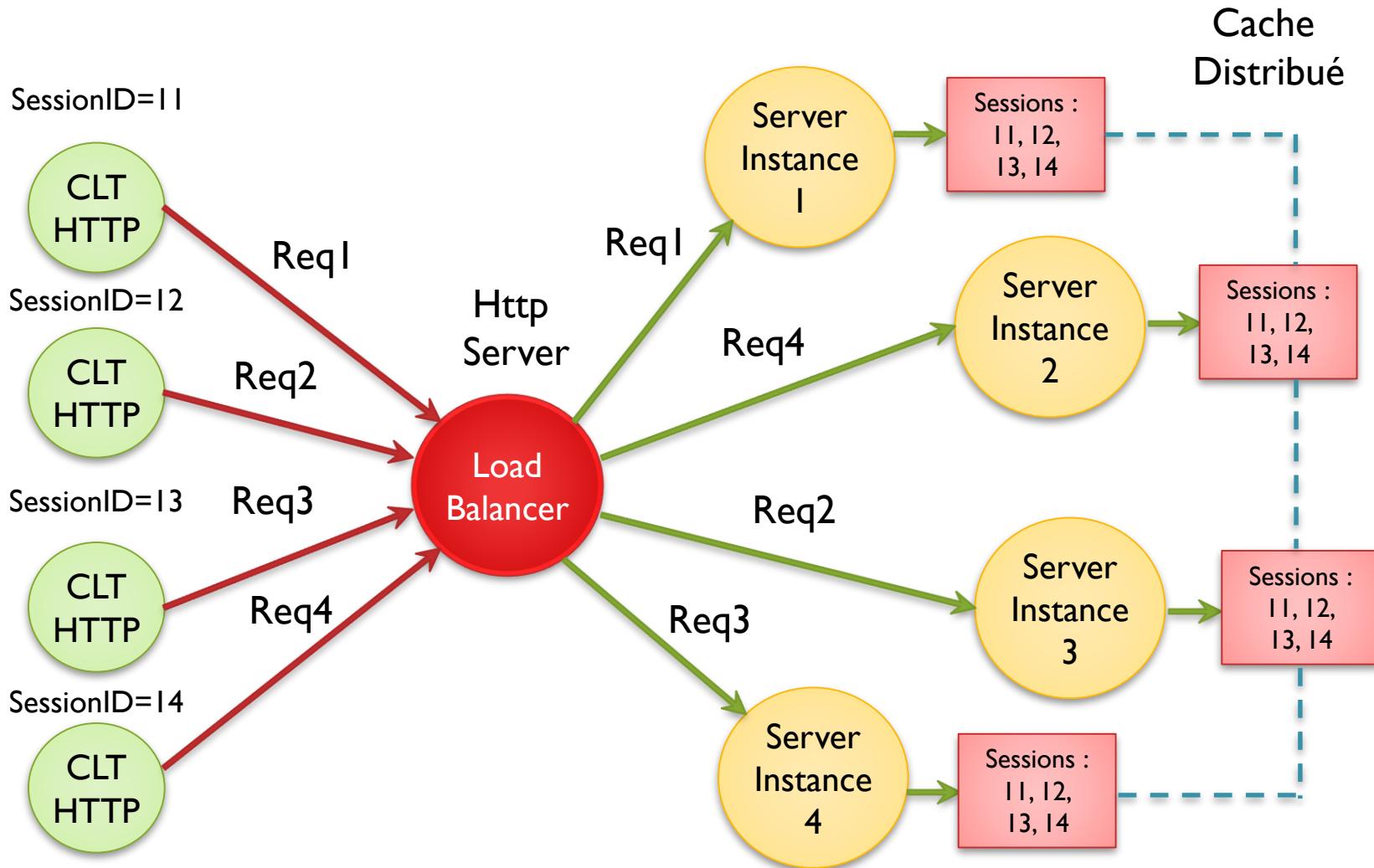
Cache Partagé pour les sessions



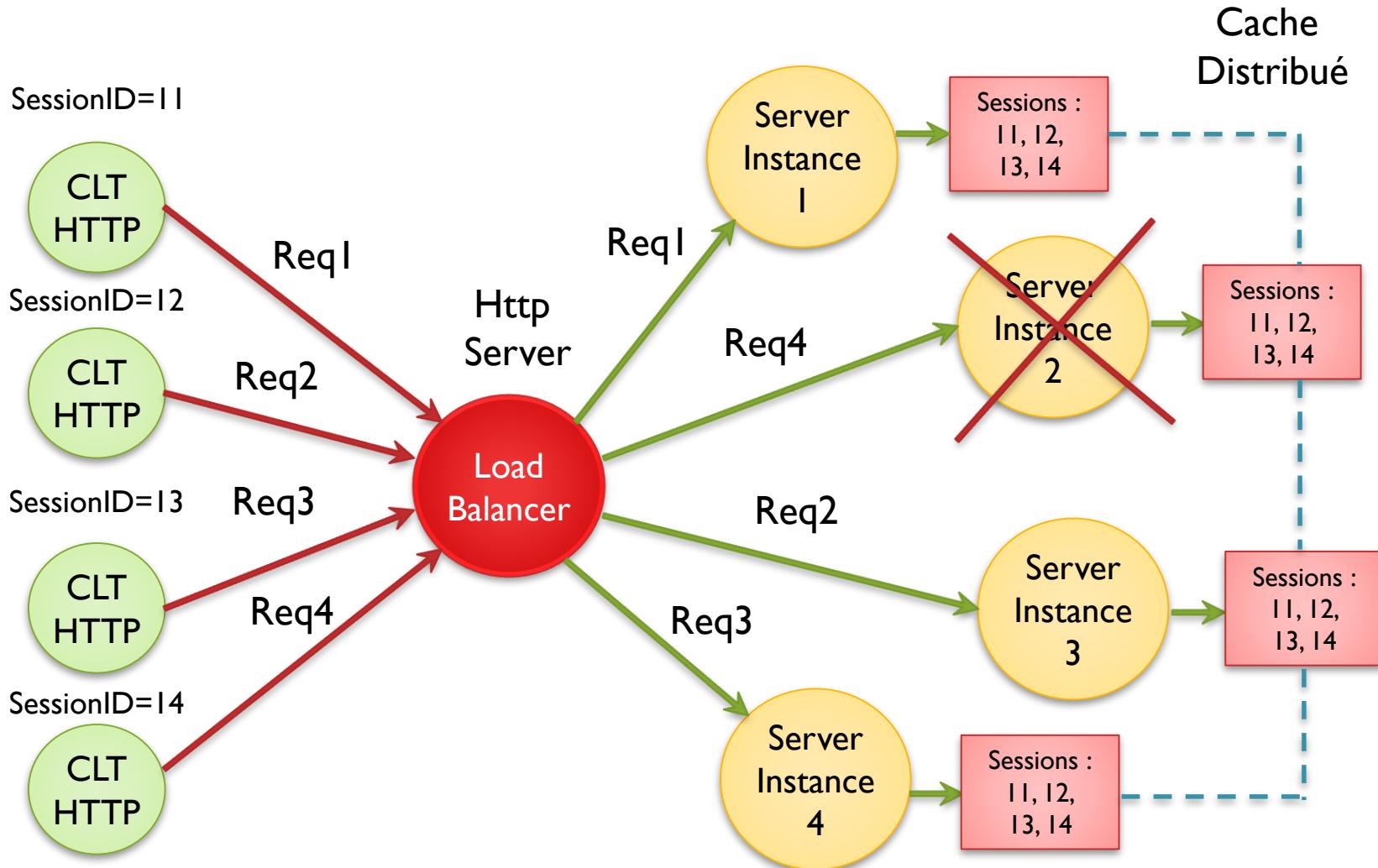
Cache Partagé pour les sessions



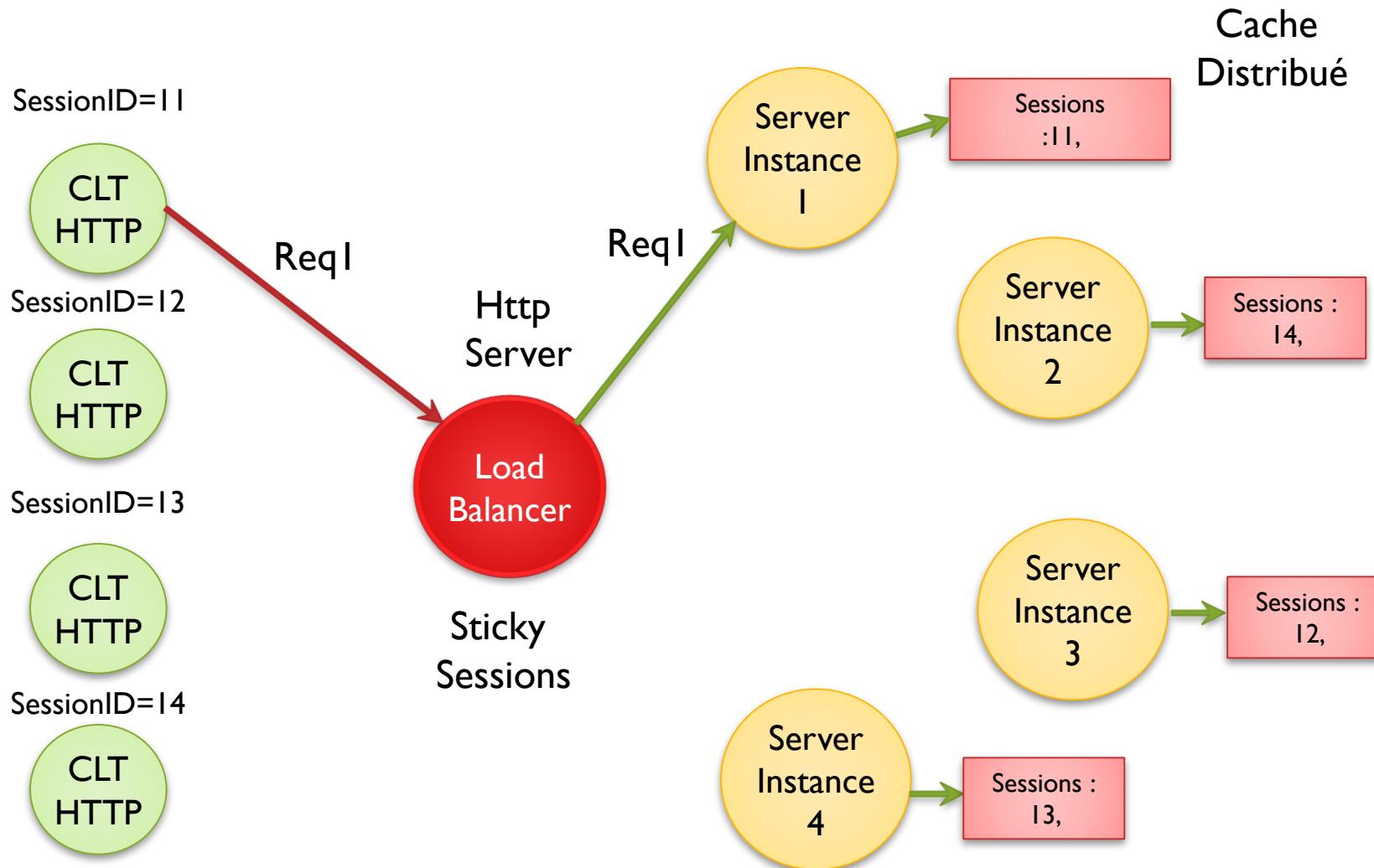
Cache Distribué pour les sessions



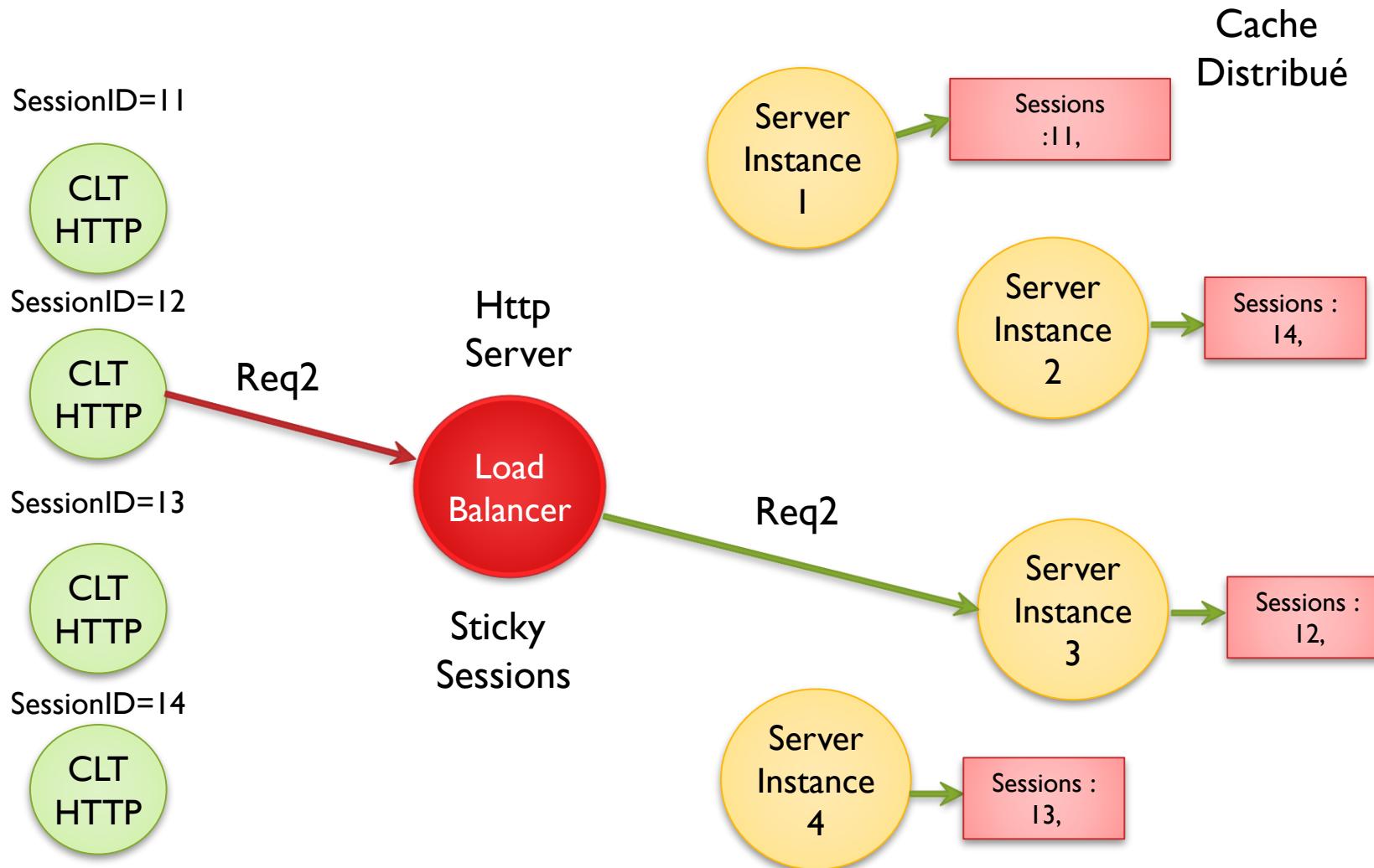
Cache Distribué pour les sessions : Tolérance aux panes



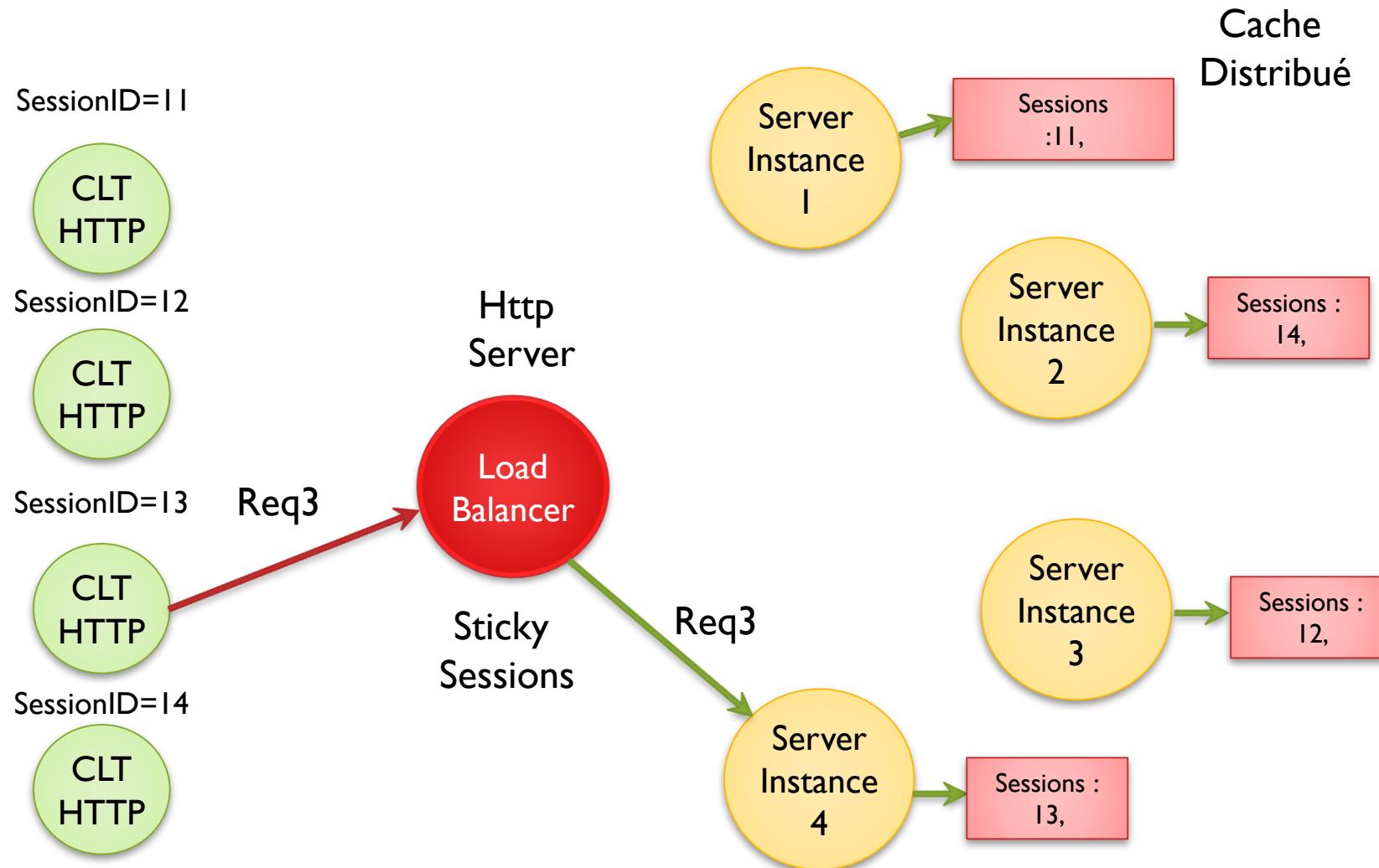
Sticky Sessions



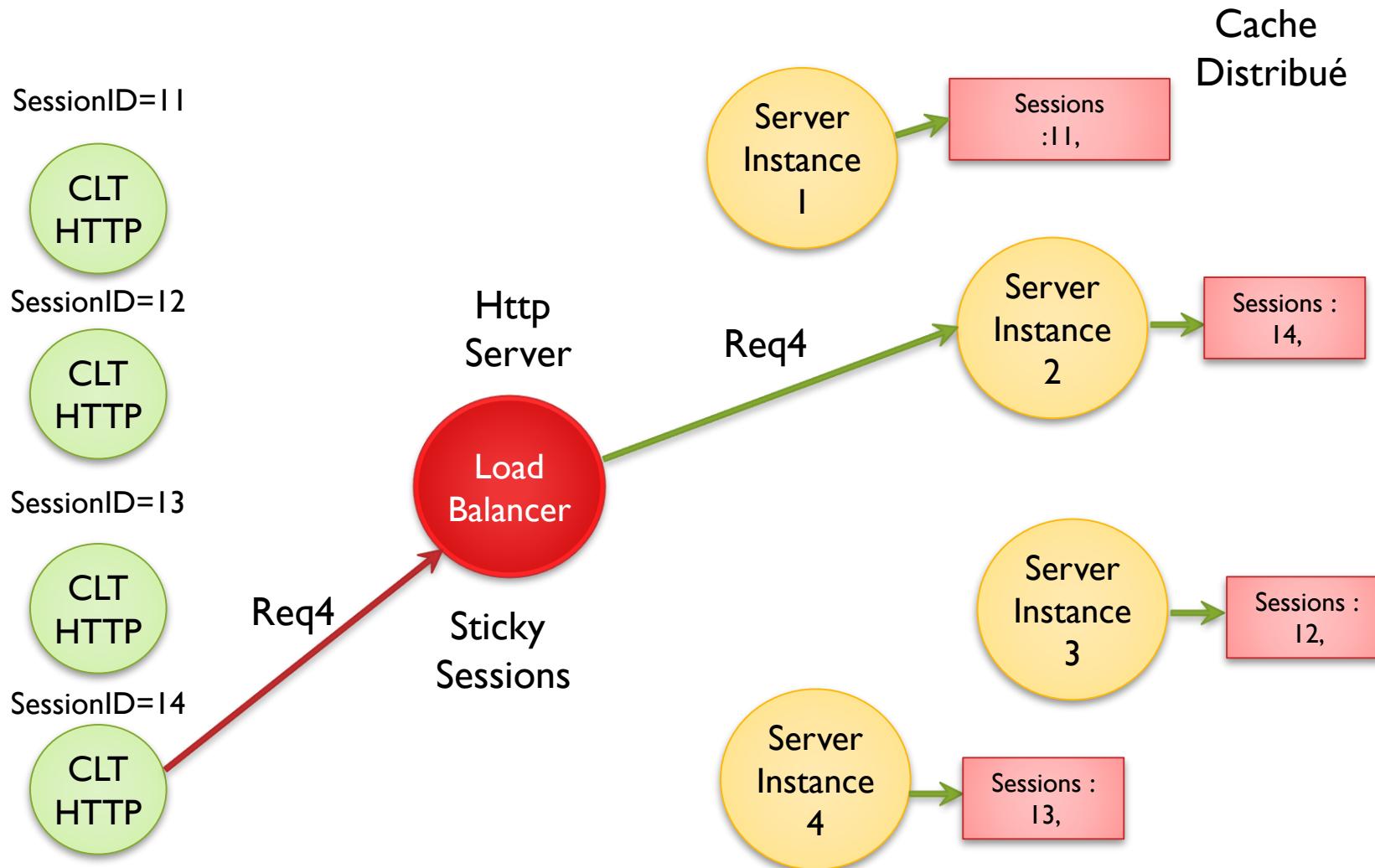
Sticky Sessions



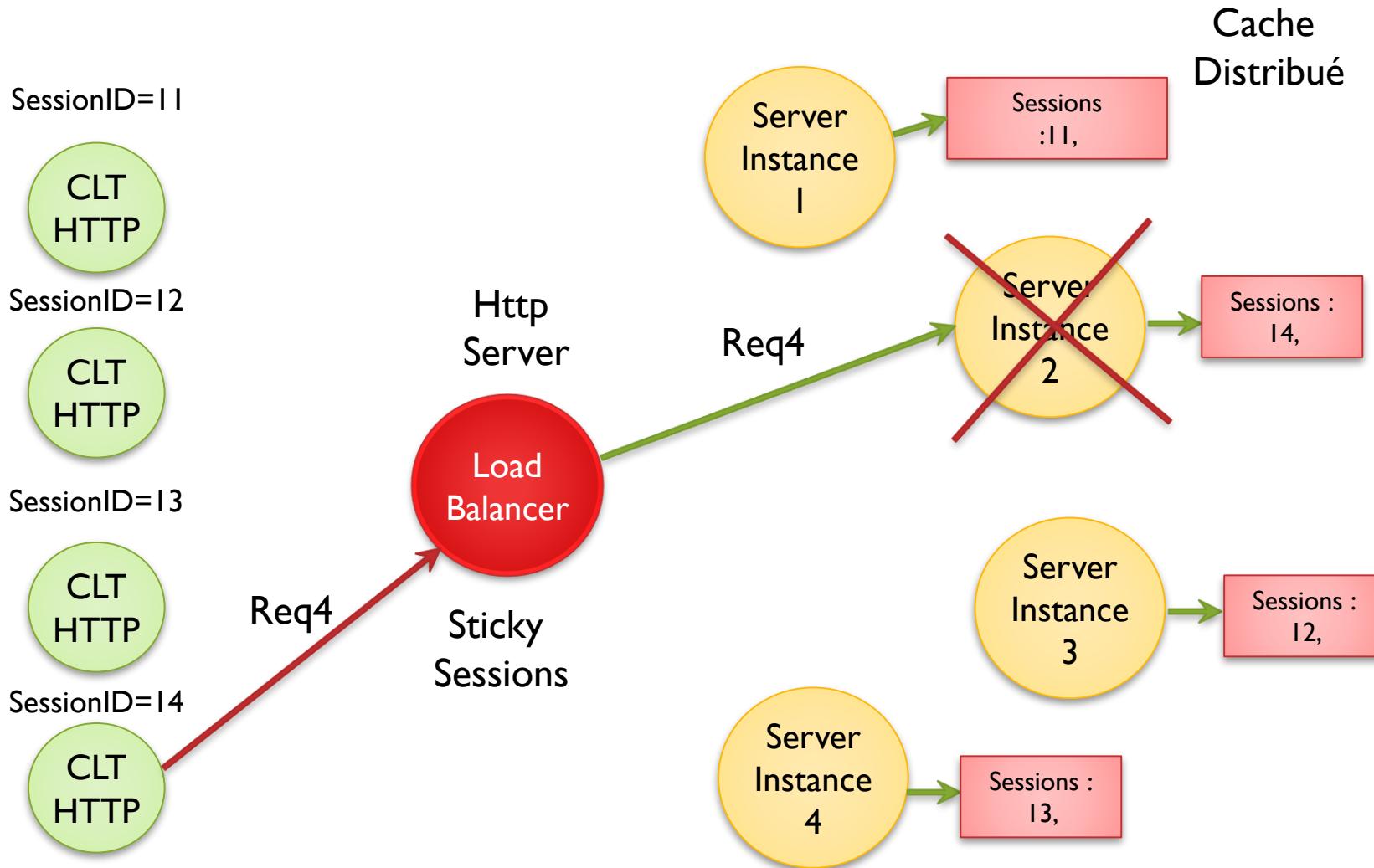
Sticky Sessions



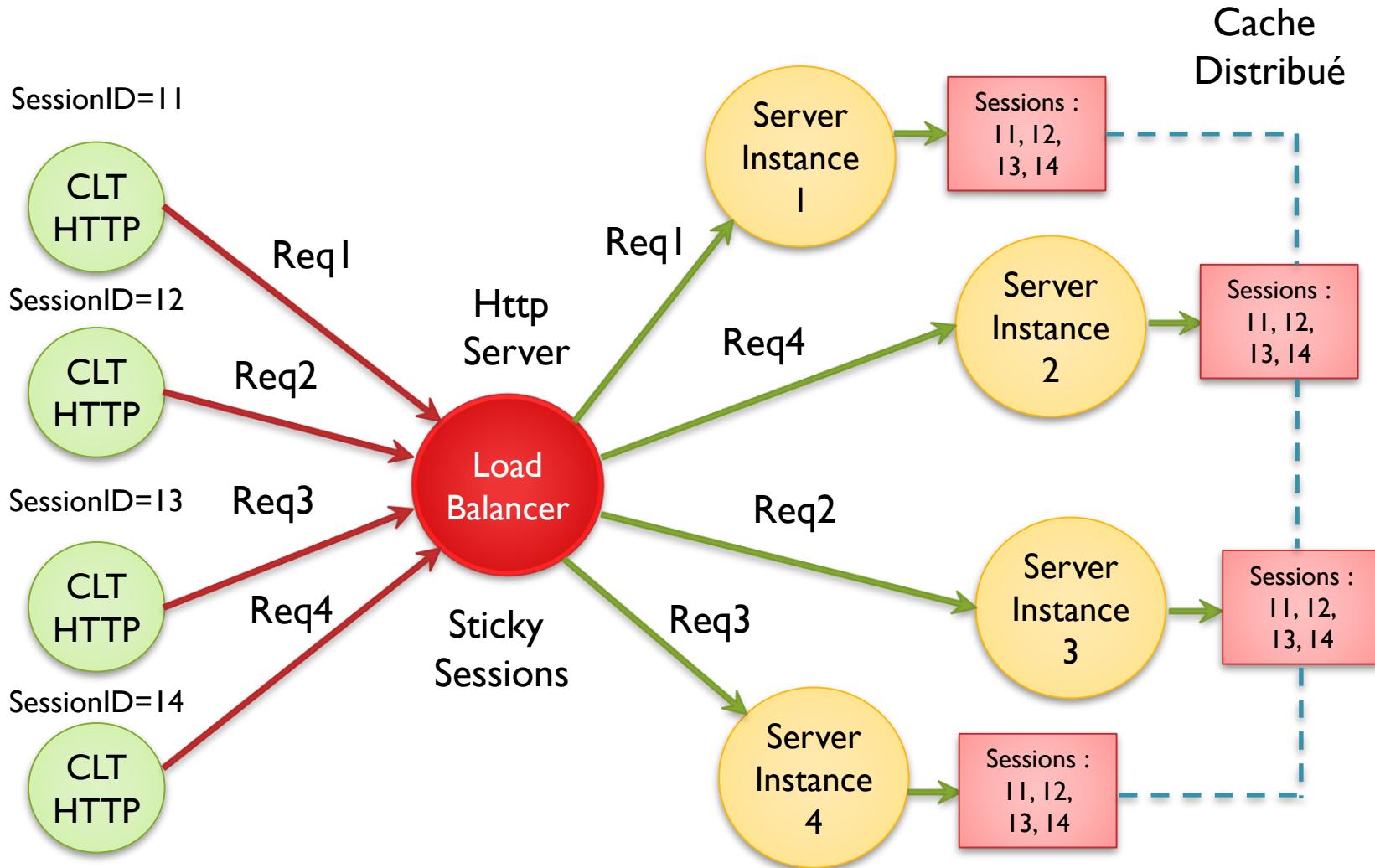
Sticky Sessions



Sticky Sessions : Problème de tolérance aux pannes



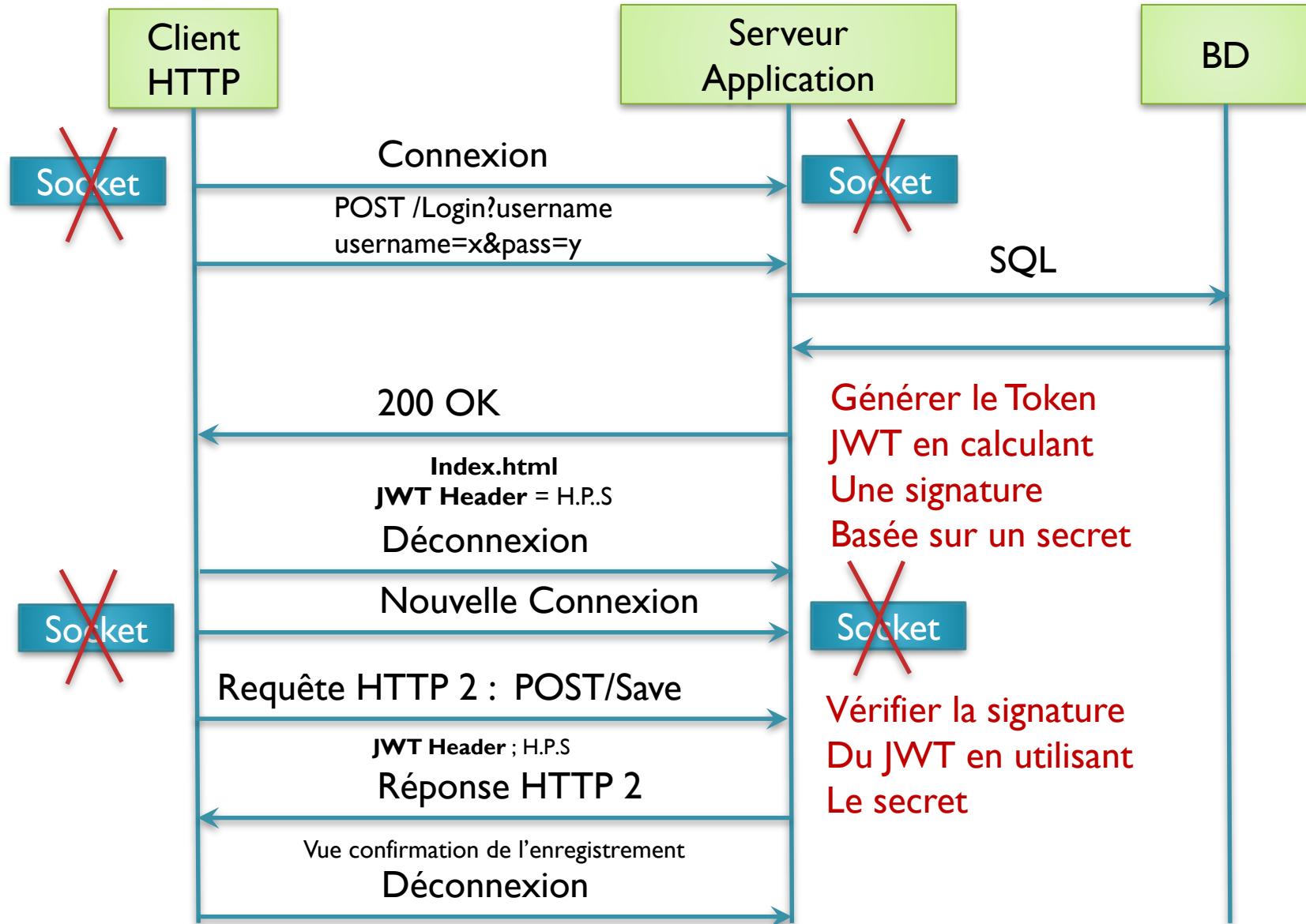
Cache Distribué et Sticky Sessions



Est-ce qu'il n'y a pas plus simple

- L'idée étant de :
 - Gérer la session côté client ?
 - Chercher un moyen qui permet au serveur de faire confiance aux clients Web
 - Sans avoir besoin de stocker sa session côté serveur

JWT



Json Web Token

- 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**.

Json Web Token

- **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 connaître le détail de l'identité d'un client authentifié.

Structure de JWT

- JWT est constitué de trois parties séparées par un point « . » :
 - Header
 - Payload
 - Signature
- La forme d'un JWT est donc :
 - **xxx.yyy.zzz**

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 privée)
 - **RSA** (**RA**di **S**hamir (2013) et **L**eonard **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 :
eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9

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 (Registered)
 - publiques (Public)
 - Privées (Private)

JWT : Payload

- **Registered 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 :

eyJzdWliOilxMjM0NTY3ODkwliwiaXNzIjoiQmFja2VuZCIsImFIZXCI6WyJXZWlgRnJvb
nQgRW5kliwiTW9iaWxIIEFcCJdLCJleHAiOjU0Nzg5MDA1LCJuYmYiOm51bGwsImI
hdCI6NDk4NjU0MzlsImP0aSI6ImIkcjU2NTQzZnR1ODkwOTg3NilsIm5hbWUiOijtZ
WQiLCJyb2xlcyI6WyJhZGlpbilslmFI dGhvcijdfQ

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

- Le JWT final est constitué des trois chaînes Base64 séparées par des points qui peuvent être facilement transmis tout en étant plus compacts.
- L'exemple suivant montre un JWT qui a l'en-tête et le Payload précédents et il est signé avec un secret.

JWT : Exemple de JWT

<https://jwt.io/>

ALGORITHM HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiQmFja2VuZCIsImF1ZCI6WyJXZWIGRnJvbnQgRW5kIiwiTW9iaWx1IEFwcCJdLCJleHAiOjU0Nzg5MDA1LCJuYmYi0m51bGwsImIhdCI6NDk4NjU0MzIsImp0aSI6ImlkcjU2NTQzZnR10Dkw0Tg3NiIsIm5hbWUi0iJtZWQiLCJyb2xlcyI6WyJhZG1pbisImF1dGhvcijdfQ.OLbtLv0UzLo4n2PQpQPb8uyE_Ut0kjq-buQLKSqlz0k
```

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

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

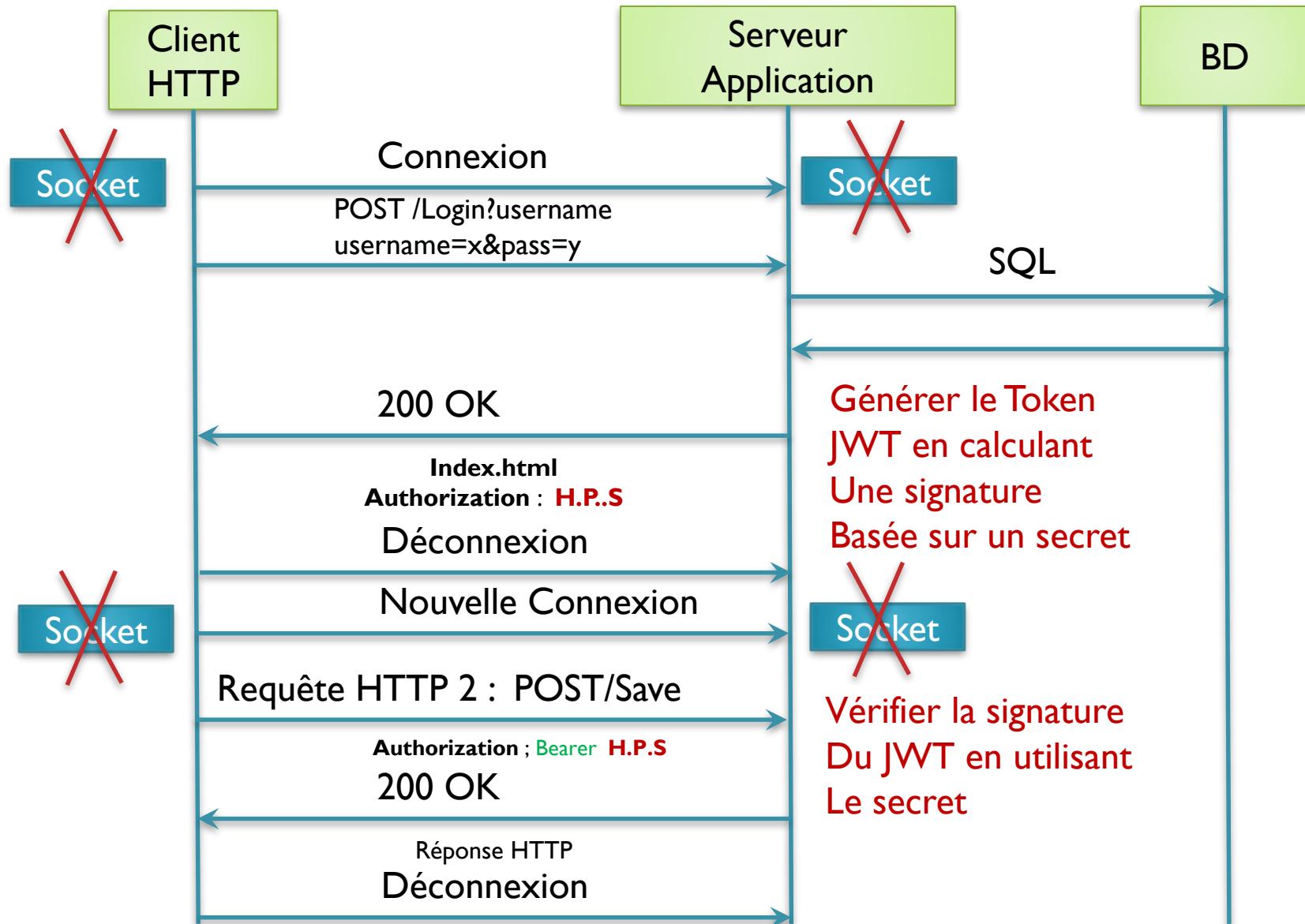
PAYOUT: DATA

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

VERIFY SIGNATURE

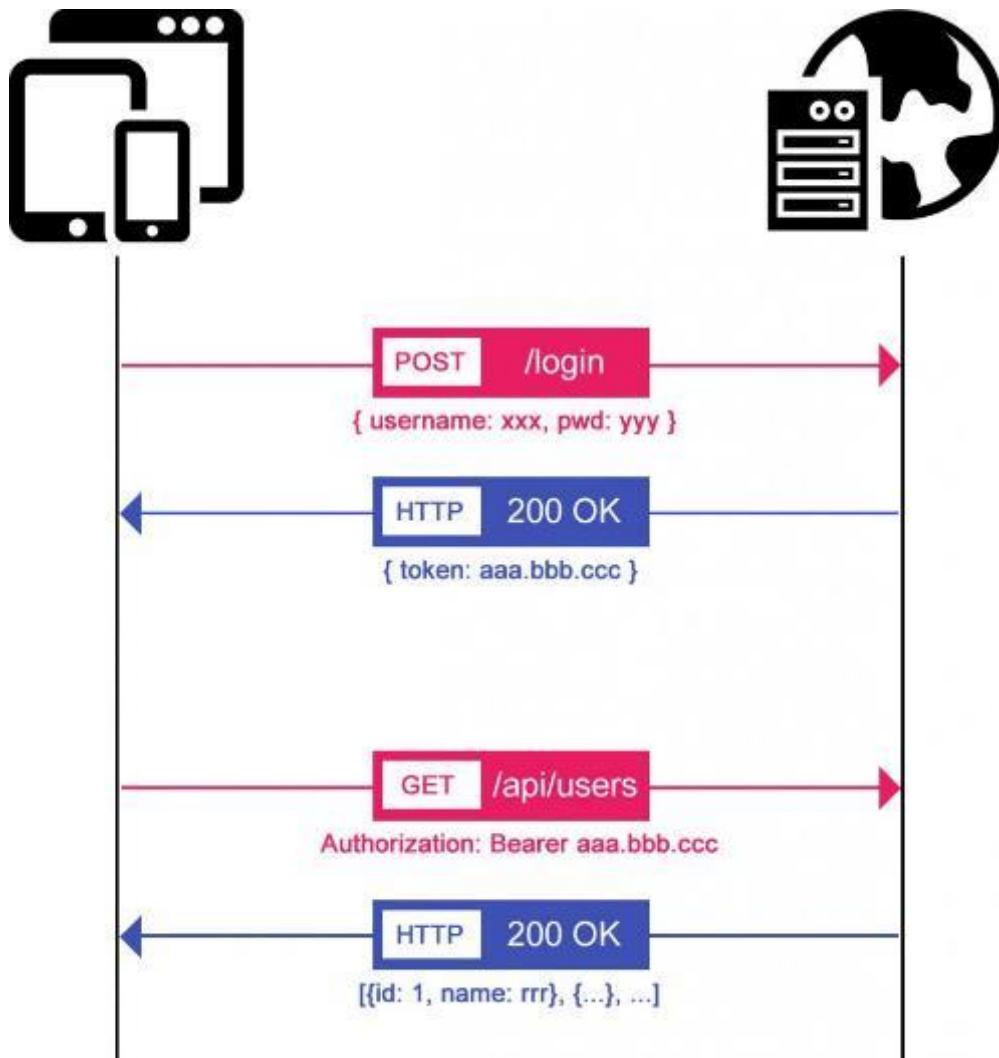
```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
)  secret base64 encoded
```

JWT

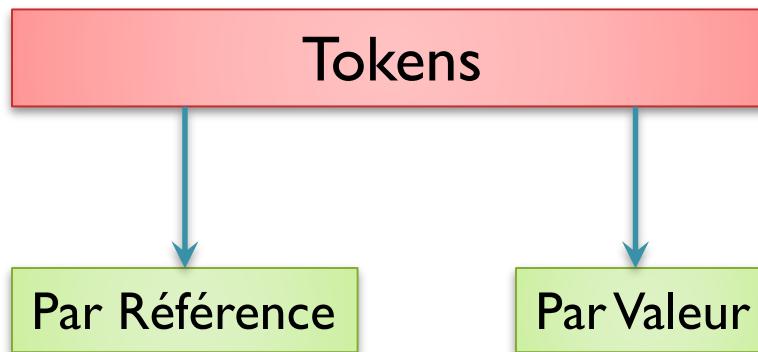


Comment utiliser JWT

- Au moment de l'authentification,
 - lorsque l'utilisateur se connecte à l'aide de ses informations d'identification,
 - un jeton Web JSON est renvoyé et doit être enregistré localement (généralement dans le stockage local, mais les cookies peuvent également être utilisés).
- Chaque fois que l'utilisateur veut accéder à une route ou à une ressource protégée, l'agent utilisateur doit envoyer le JWT, généralement dans l'en-tête Authorization en utilisant le schéma Bearer. Le contenu de l'en-tête devrait ressembler à ceci:
 - **Authorization: Bearer <token>**



Session ID Vs JWT



Session ID = 13

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

JWT

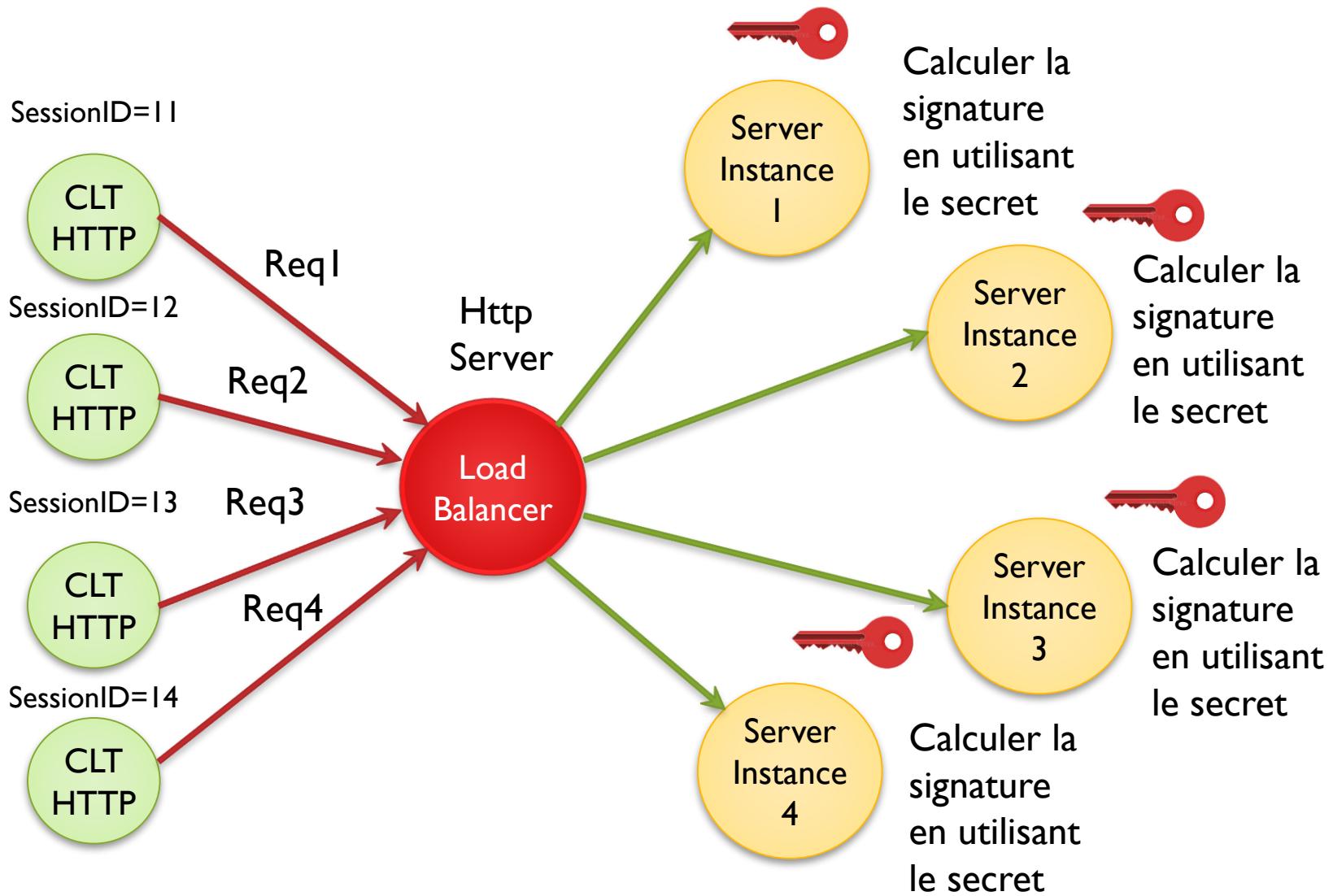
- N'a pas besoin d'une application tiers pour connaître les informations associés à ce Token
- Le JWT contient lui-même toutes les informations sur l'utilisateur d'il prétend représenter

Avantages de JWT

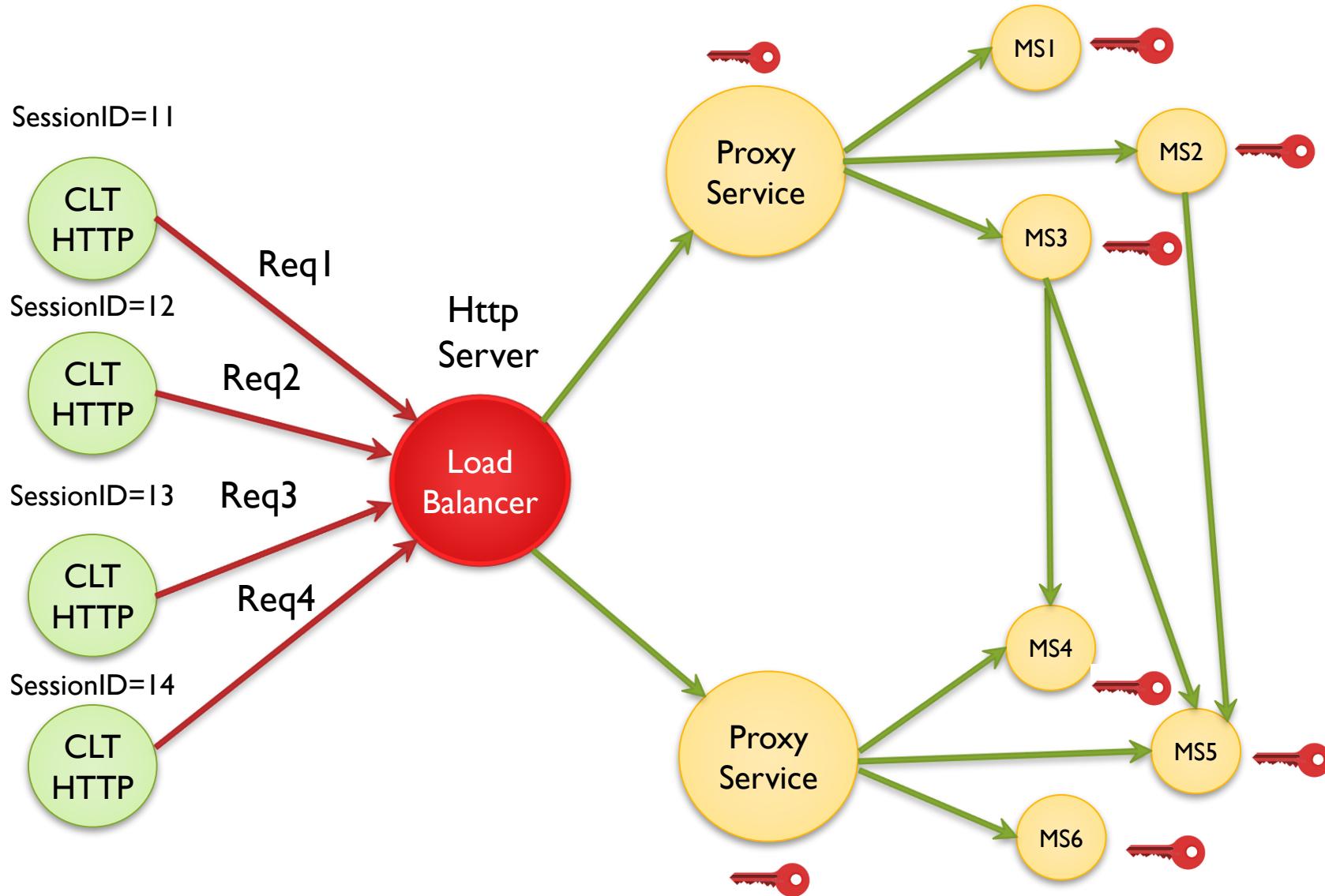
Implémentations multi langages

- Java
- PHP
- Java Script
- Python
- .Net
- Etc

Problème de montée en charge. Pas de besoin de cache partagé ou distribué



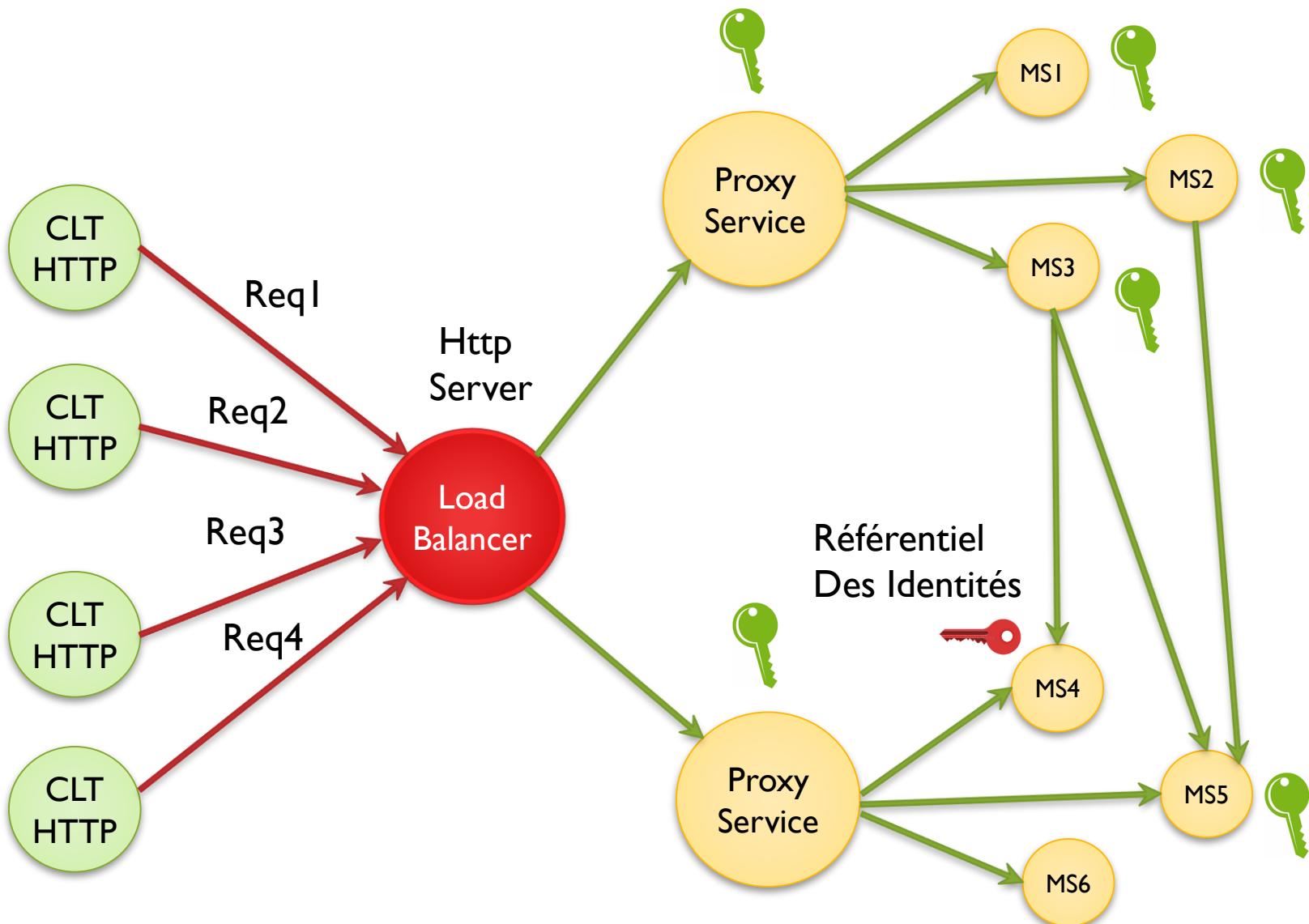
Bonne solution pour les architectures distribuées basées sur les micro services



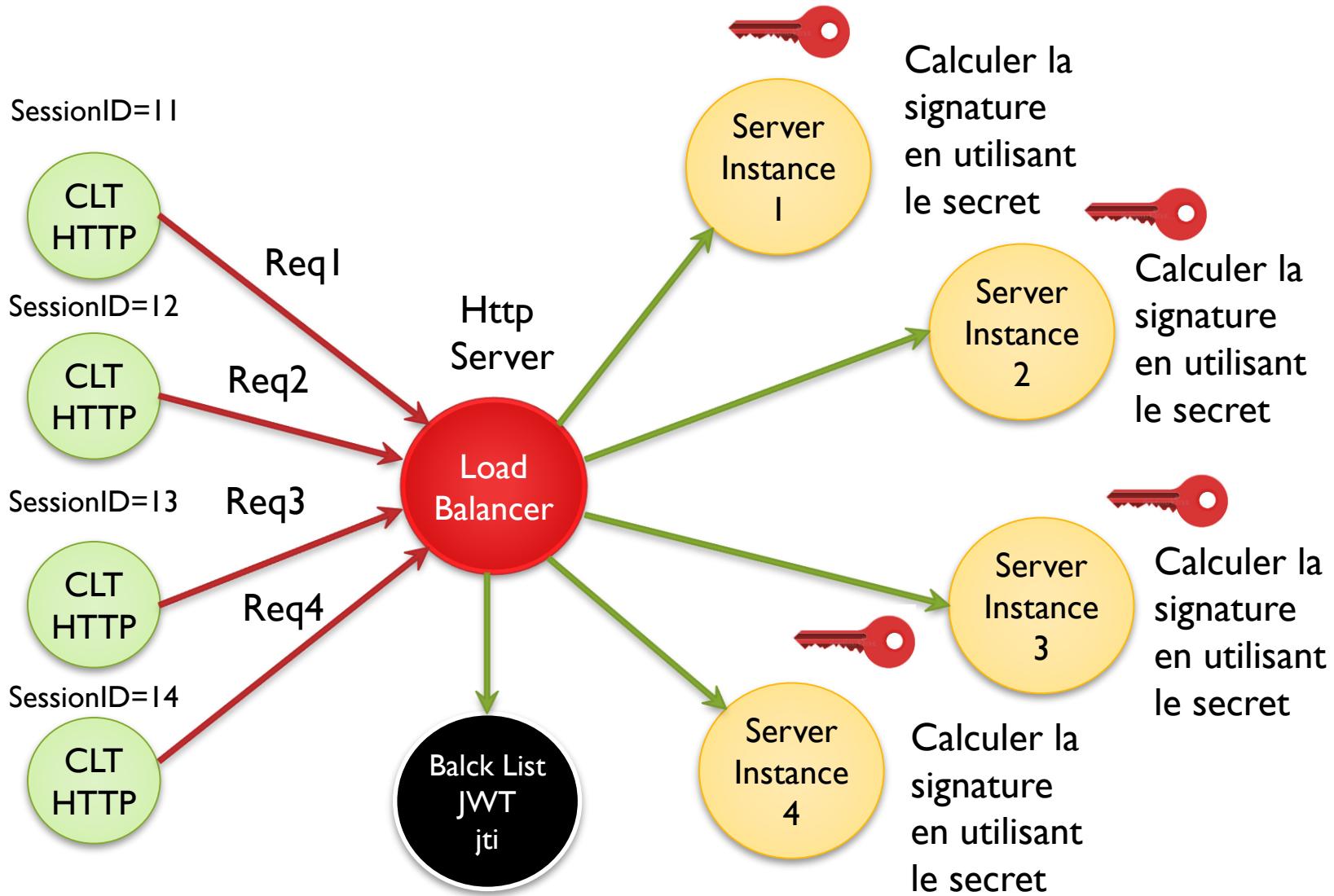
Secret partagé?

- Il existe deux manière de signer le JWT :
 - Symétrique (Alg=HS256) avec
 - un secret partagé
 - Asymétrique (Alg=RS256) avec
 - une clé privée : pour signer et générer les JWT
 - une clé publique : pour vérifier la validé des JWT

JWT dans un système distribué avec clé privée et clé publique



Révocation des Tokens



Où stocker le JWT

- **Jeton stocké dans le sessionStorage ou dans le localStorage du navigateur :**
 - Si le jeton est stocké de cette manière, il devra être inclus systématiquement dans les requêtes envoyées au serveur, par exemple via le header « Authorization: Bearer <jeton> ».
 - Cette solution est donc idéale pour des applications principalement Frontend en Javascript et où celui-ci fonctionne principalement avec des requêtes AJAX.
 - Le développeur devra donc ajouter le jeton dans chaque requête.
 - Cette solution a l'avantage de protéger nativement l'application contre les attaques de type CSRF.
 - Cependant, comme le jeton doit être disponible pour l'application JavaScript, il sera exposé en cas de faille XSS et pourra être récupéré.

Où stocker le JWT

- **Jeton stocké dans des cookies :**
 - Lorsqu'un jeton JWT est stocké dans un cookie sur le navigateur de l'utilisateur, il est faudrait penser de lui attribuer les flags “**HttpOnly**” (et “**Secure**”) afin de le protéger contre une éventuelle récupération via une attaque de type XSS.
 - Cette solution a l'inconvénient de ne pas permettre une authentification sur une autre application d'un autre domaine car les cookies, par mesure de sécurité, ne peuvent être renvoyés que sur le domaine du serveur qui les a créés.
 - Avec un jeton stocké dans un cookie, celui-ci étant automatiquement envoyé par le navigateur avec chaque requête. Ce qui l'expose aux attaques **CSRF**
 - Pour une prévention contre les attaques CSRF, l'une des solutions c'est d'inclure le CSRF Synchronized Token dans les claims du JWT et resigner le JWT pour chaque réponse HTTP.

Sécuriser les JWT

- Pour les Applications Web de type Single page Application :
 - Il faut faire attention aux attaques XSS (Cross Site Scripting) : Injection des contenus utilisateurs
 - Tous les points d'accès qui ne sont pas Https peuvent exécuter du java script et prendre le contenu des requêtes HTTP.
 - Si vous utilisez le cookies pour stocker le JWT, penser à envoyer le JWT en mode :
 - Secure : Se passe qu'avec HTTPS
 - HTTPOnly : Ne pas permettre à java script de lire le contenu du cookie

200 OK

Set-Cookie : jwt=header.payload.signature;
Secure;HTTPOnly

Sécuriser les JWT

- Si vous stockez les JWT dans des local storages ou cookies non sécurisé, n'importe quel script peut voler le Token et le balancer ailleurs.
- Après, ils peuvent s'en servir pour se faire passer pour vous (Fausse identité)

Sécuriser les JWT

- Pour les applications mobile et serveurs tiers, il faudrait envoyer le JWT dans les entêtes HTTP

200 OK

Autorisation : Bearer **Header.Payload.Signature**

Sécuriser les JWT

- Pour les attaques de type CSRF (Cross Site Request Forgery)
 - Pensez à mettre le Synchronizer token (CSRF Authenticity-ID) dans le Payload du JWT.

Autres usages des JWT

- Formulaires en plusieurs parties
 - On peut stocker, côté serveur, les données saisies dans la première page dans le JWT et l'envoyer au client dans la page 2, jusqu'à la page finale qui sera stockée côté serveur.
- Confirmation des email
 - Comme le JWT est en base64URL, on peut générer une JWT avec une date d'expiration réduite (2 min par exemple) et l'envoyer par mail sous forme d'une URL
 - Ce qui permet d'éviter du stockage côté serveur pour accomplir cette opération.

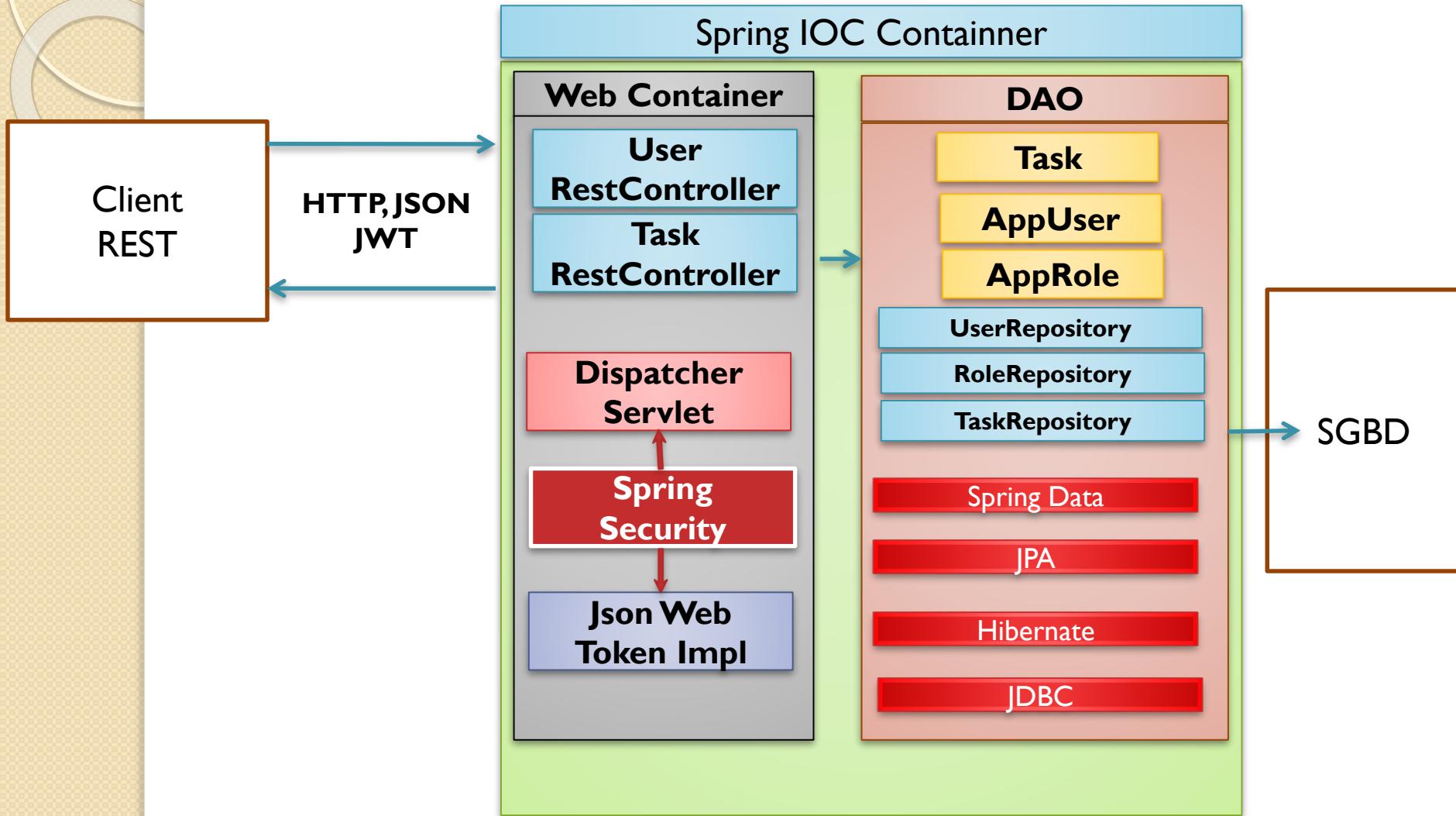
Application :

JWT et Spring Security

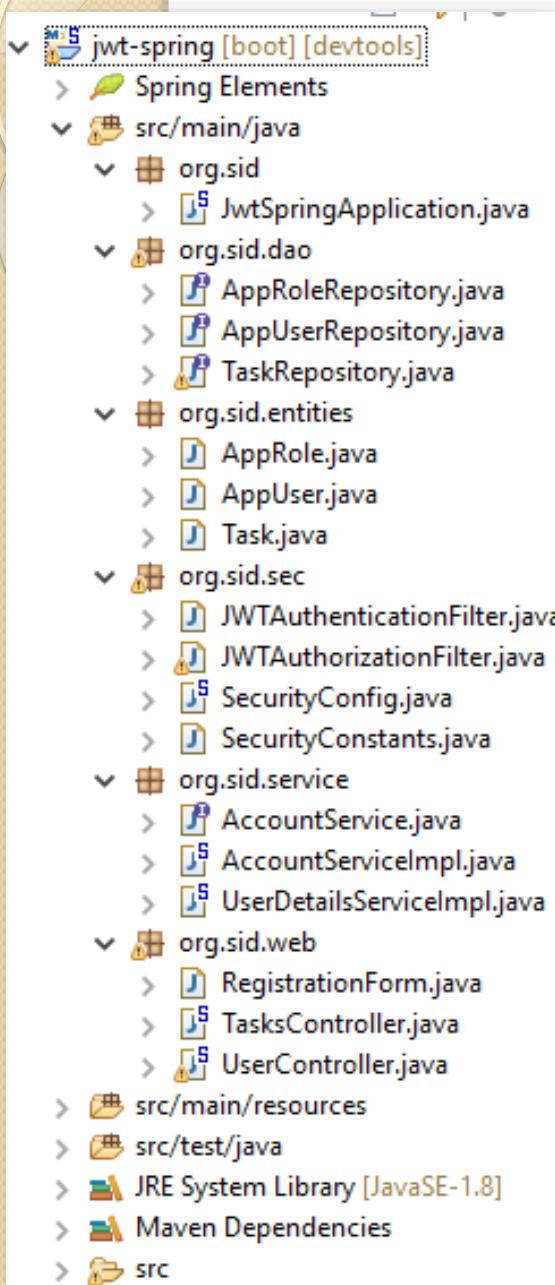
Spring Security et JWT

- Créer une application qui permet de gérer des tâches.
- L'accès à l'API REST est sécurisé d'une manière statless, par Spring security en utilisant Json Web Token

Architecture

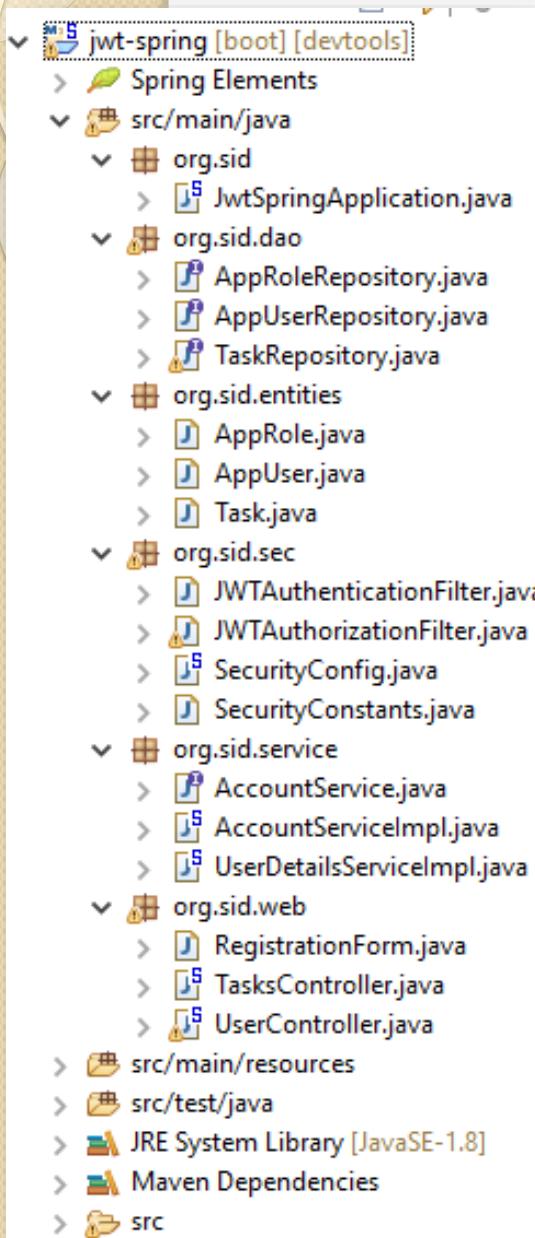


Structure du Projet



```
<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>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.7.0</version>
    </dependency>
```

Structure du Projet



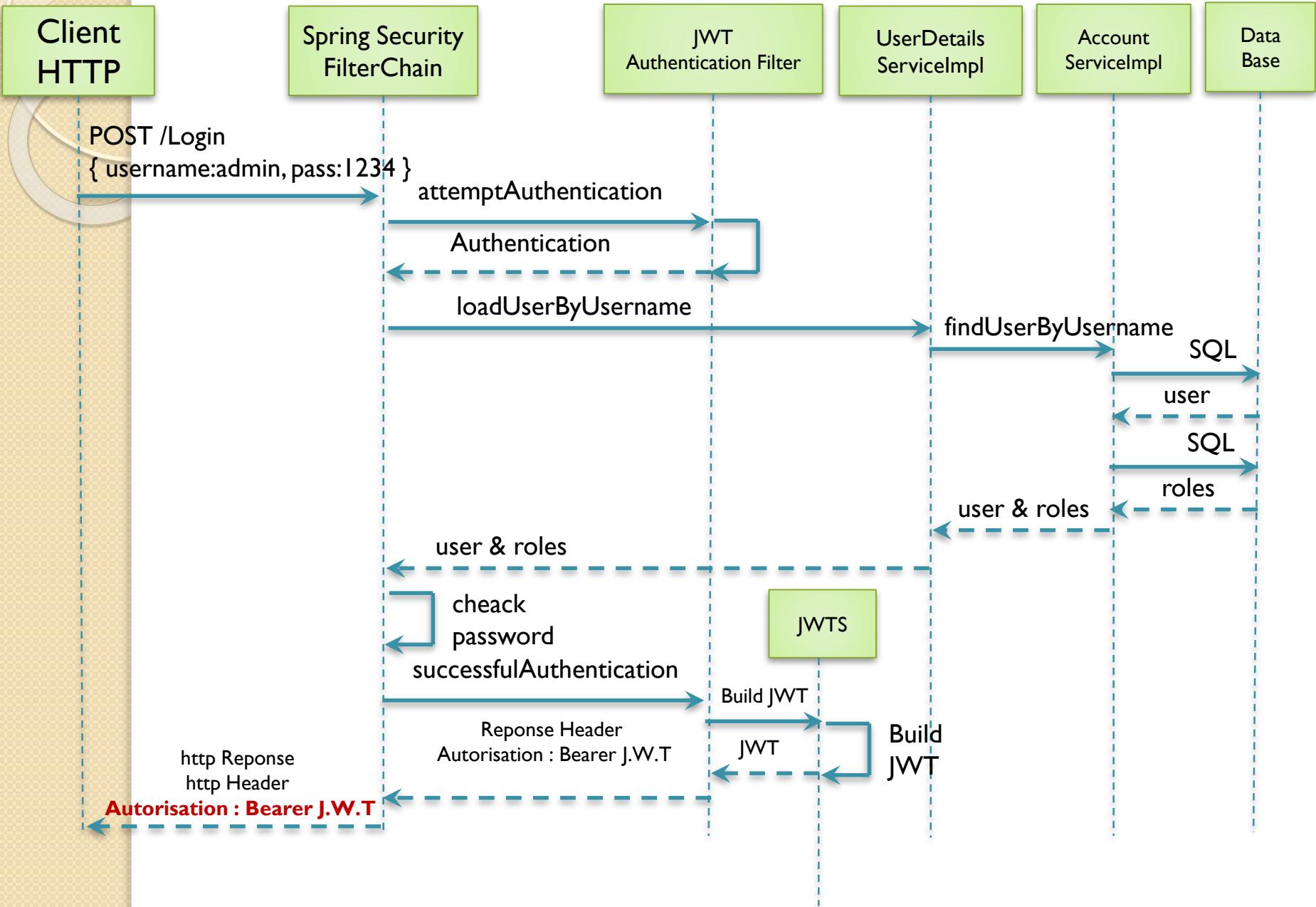
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>

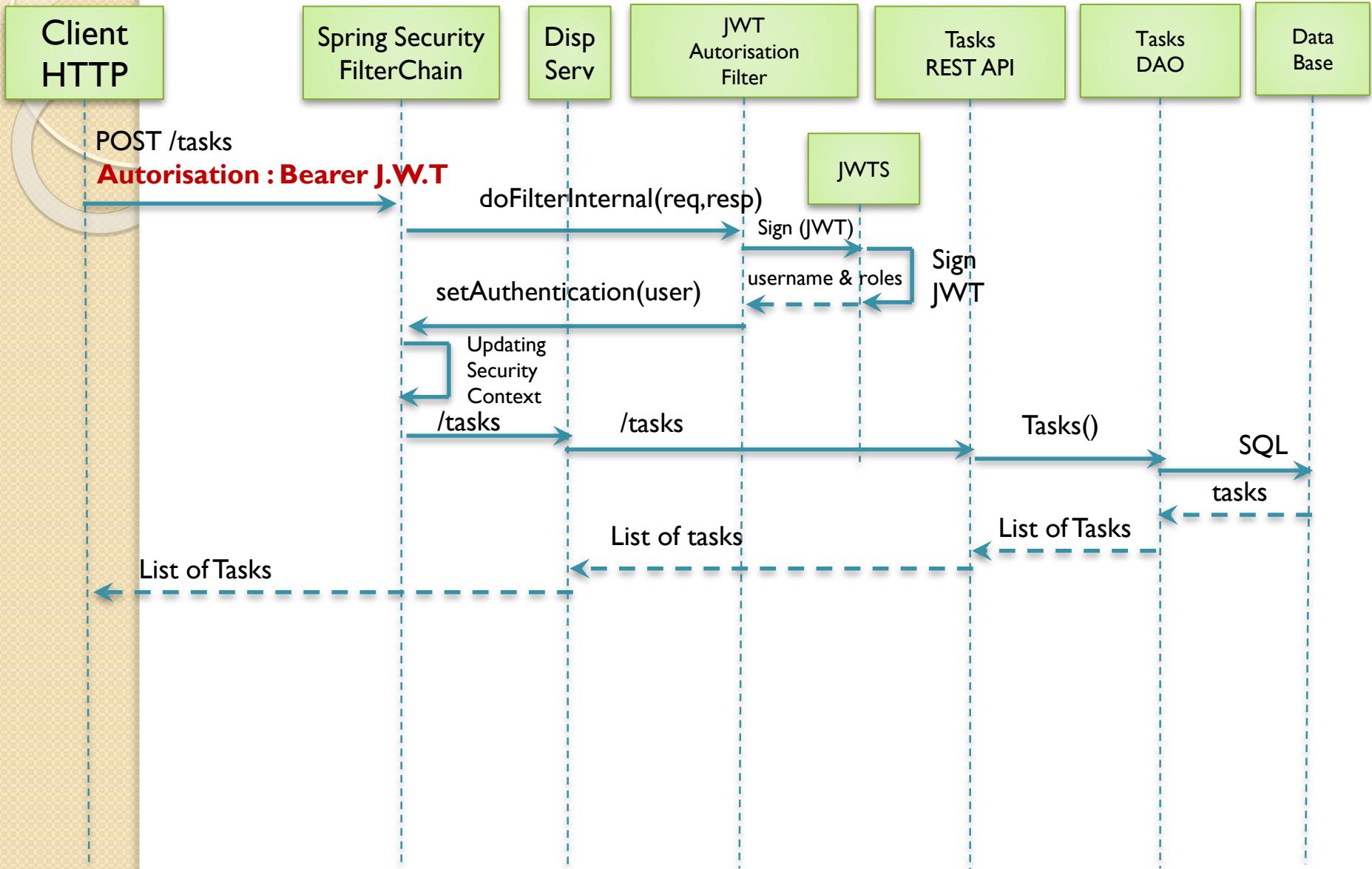
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

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

Authentification Spring Security avec JWT



Demander une ressource nécessitant l'authentification



Partie FrontEnd Web avec Angular 5

The image shows three browser tabs illustrating the Angular 5 application:

- localhost:4200/login**: The login page features a "Login" button and a "Register New User" button.
- localhost:4200/tasks**: The task list page displays three tasks: T1, T2, and T3.
- localhost:4200/new-task**: The new task creation page shows a "Task Name" input field containing "JWT" and a "Save" button.

The image shows a browser window with the following components:

- localhost:4200/new-task**: Shows a success message: "New task Added" with ID:4 and Task Name:JWT.
- localhost:4200/tasks**: Shows the updated task list with four entries: T1, T2, T3, and JWT.
- Developer Tools (Application tab)**: Shows the browser storage with a token entry:

Key	Value
token	Bearer eyJhbGciOiJIUzUxMiJ9eyJzdWliOiJhZG1pbilsIn

Partie FrontEnd Web avec Angular 5

The image displays a screenshot of a web application built with Angular 5, showing multiple browser tabs and windows.

- localhost:4200/login:** Shows an authentication form with "Username" (user) and "Password" (redacted). Buttons for "Login" (red) and "Register New User" (green).
- localhost:4200/tasks:** Shows a table of tasks with columns "ID" and "Task Name". Data: ID 1 (T1), ID 2 (T2), ID 3 (T3), ID 4 (JWT).
- localhost:4200/register:** Shows a registration form with "Username" (hassan), "Password" (redacted), and "Confirm Password" (redacted). A red message box says "You must confirm your password".
- localhost:4200/register (bottom-left):** Shows a registration form with "Username" (mohamed), "Password" (redacted), and "Confirm Password" (redacted). A red message box says "This user already exists, Try with an other username".
- localhost:4200/register (bottom-right):** Shows a registration form with "Username" (mohamed), "Password" (redacted), and "Confirm Password" (redacted). A red "Register" button is visible.
- localhost:4200/register (center):** Shows a "Registration Success" message with "Your ID :3", "Your username :mohamed", and "Your role:USER".

Entités JPA

```
package org.sid.entities;
import javax.persistence.*;
import lombok.*;
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Task {
    @Id @GeneratedValue
    private Long id;
    private String taskName;
}
```

```
package org.sid.entities;
import javax.persistence.*;
import lombok.*;
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class AppRole {
    @Id @GeneratedValue
    private Long id;
    private String role;
}
```

```
package org.sid.entities;
import java.util.*;
import javax.persistence.*;
import lombok.*;
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class AppUser {
    @Id @GeneratedValue
    private Long id;
    private String username;
    private String password;
    @ManyToMany(fetch=FetchType.EAGER)
    private Collection<AppRole> roles=new ArrayList<>();
}
```

Couche DAO basée sur Spring Data

```
package org.sid.dao;  
import org.sid.entities.Task;  
import org.springframework.data.jpa.repository.JpaRepository;  
public interface TaskRepository extends JpaRepository<Task, Long> {  
}
```

```
package org.sid.dao;  
import org.sid.entities.AppUser;  
import org.springframework.data.jpa.repository.JpaRepository;  
public interface AppUserRepository extends JpaRepository<AppUser, Long> {  
    public AppUser findByUsername(String username);  
}
```

```
package org.sid.dao;  
import org.sid.entities.AppRole;  
import org.springframework.data.jpa.repository.JpaRepository;  
public interface AppRoleRepository extends JpaRepository<AppRole, Long> {  
    public AppRole findByName(String role);  
}
```

Couche Service pour la gestion des utilisateurs

```
package org.sid.service;

import org.sid.entities.AppRole; import org.sid.entities.AppUser;

public interface AccountService {

    public AppUser saveUser(AppUser u);

    public AppRole saveRole(AppRole r);

    public AppUser findUserByUsername(String username);

    public void addRoleToUser(String username, String role);

}
```

```
package org.sid.service;

import javax.transaction.Transactional; import org.sid.dao.*; import org.sid.entities.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

@Service
@Transactional
public class AccountServiceImpl implements AccountService {

    @Autowired
    private AppUserRepository userRepository;
    @Autowired
    private AppRoleRepository roleRepository;
    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;
```

Couche Service pour la gestion des utilisateurs

```
@Override  
public AppUser saveUser(AppUser u) {  
    u.setPassword(bCryptPasswordEncoder.encode(u.getPassword()));  
    return userRepository.save(u);  
}  
  
@Override  
public AppRole saveRole(AppRole r) {  
    return roleRepository.save(r);  
}  
  
@Override  
public AppUser findUserByUsername(String username) {  
    return userRepository.findByUsername(username);  
}  
  
@Override  
public void addRoleToUser(String username, String roleName) {  
    AppUser user=userRepository.findByUsername(username);  
    AppRole role=roleRepository.findByRole(roleName);  
    user.getRoles().add(role);  
}  
}
```

Couche Web : API REST pour gérer les taches

```
package org.sid.web;

import java.util.List; import org.sid.dao.TaskRepository;

import org.sid.entities.Task; import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody; import org.springframework.web.bind.annotation.RestController;
```

@RestController

```
public class TasksController {

    @Autowired
    private TaskRepository taskRepository;

    @GetMapping("/tasks")
    public List<Task> listTasks(){
        return taskRepository.findAll();
    }

    @PostMapping("/tasks")
    public Task save(@RequestBody Task task){
        return taskRepository.save(task);
    }
}
```

Couche Web : API REST pour enregistrer de nouveaux utilisateurs

```
package org.sid.web;

import org.sid.entities.AppUser; import org.sid.service.AccountService; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping; import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

    @Autowired
    private AccountService accountService;

    @PostMapping("/users")
    public AppUser signUp(@RequestBody RegistrationForm data) {
        String username=data.getUsername();
        AppUser user=accountService.findUserByUsername(username);
        if(user!=null) throw new RuntimeException("This user already exists, Try with an other
username");
        String password=data.getPassword();    String repassword=data.getRepassword();
        if(!password.equals(repassword))
            throw new RuntimeException("You must confirm your password");
        AppUser u=new AppUser();  u.setUsername(username);  u.setPassword(password);
        accountService.saveUser(u);
        accountService.addRoleToUser(username, "USER");
        return (u);
    }
}
```

RegistrationForm.java

```
package org.sid.web;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data@AllArgsConstructor@NoArgsConstructor
public class RegistrationForm {
    private String username;
    private String password;
    private String repassword;
}
```

Configuration de Spring Security

```
package org.sid.sec;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService)
            .passwordEncoder(bCryptPasswordEncoder);
    }
}
```

Configuration de Spring Security

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
  
    http.csrf().disable()  
    // don't create session  
  
    .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
    .and()  
        .authorizeRequests()  
        .antMatchers("/users/**", "/login/**")  
        .permitAll()  
        .antMatchers(HttpMethod.POST, "/tasks/**").hasAuthority("ADMIN")  
        .anyRequest().authenticated()  
    .and()  
        .addFilter(new JWTAuthenticationFilter(authenticationManager()))  
        .addFilterBefore(new JWTAuthorizationFilter(),  
UsernamePasswordAuthenticationFilter.class);  
}
```

Constantes de l'application

```
package org.sid.sec;

public class SecurityConstants {
    public static final String SECRET = "med@youssf.net";
    public static final long EXPIRATION_TIME = 864_000_000; // 10 days
    public static final String TOKEN_PREFIX = "Bearer ";
    public static final String HEADER_STRING = "Authorization";
}
```

Couche Service : pour la gestion des utilisateurs

```
package org.sid.service;

import java.util.ArrayList;import java.util.Collection;

import org.sid.entities.AppUser; import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.User; import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

@Service

public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired

    private AccountService accountService;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

    AppUser u=accountService.findUserByUsername(username);

    if(u==null) throw new UsernameNotFoundException(username);

    Collection<GrantedAuthority> authorities=new ArrayList<>();

    u.getRoles().forEach(r->{

        authorities.add(new SimpleGrantedAuthority(r.getRole()));

    });

    return new User(u.getUsername(), u.getPassword(), authorities);

}

}
```

JWTAuthenticationFilter

```
package org.sid.sec;

import java.io.IOException; import java.util.Date; import javax.servlet.FilterChain;
import javax.servlet.ServletException; import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse; import org.sid.entities.AppUser;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import com.fasterxml.jackson.databind.ObjectMapper; import io.jsonwebtoken.Jwts; import io.jsonwebtoken.SignatureAlgorithm;

public class JWTAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
    private AuthenticationManager authenticationManager;

    public JWTAuthenticationFilter(AuthenticationManager authenticationManager) {
        super();
        this.authenticationManager = authenticationManager;
    }
}
```

JWTAuthenticationFilter

```
@Override  
  
public Authentication attemptAuthentication(HttpServletRequest request,  
HttpServletResponse response) throws AuthenticationException {  
  
    AppUser user=null;  
    try {  
        user = new ObjectMapper().readValue(request.getInputStream(), AppUser.class);  
    } catch (Exception e) {  
        throw new RuntimeException(e);  
    }  
    return authenticationManager.authenticate(  
        new UsernamePasswordAuthenticationToken(  
            user.getUsername(),  
            user.getPassword()  
        ));  
}
```

JWTAuthenticationFilter

```
@Override  
  
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,  
Authentication authResult) throws IOException, ServletException {  
  
    User springUser=(User)authResult.getPrincipal();  
    String jwtToken=Jwts.builder()  
        .setSubject(springUser.getUsername())  
        .setExpiration(new  
Date(System.currentTimeMillis()+SecurityConstants.EXPIRATION_TIME))  
        .signWith(SignatureAlgorithm.HS512, SecurityConstants.SECRET)  
        .claim("roles", springUser.getAuthorities())  
        .compact();  
    response.addHeader(SecurityConstants.HEADER_STRING,  
SecurityConstants.TOKEN_PREFIX+jwtToken);  
}  
}
```

JWTAuthorizationFilter

```
package org.sid.sec;

import java.io.IOException; import java.util.*; import javax.servlet.FilterChain; import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.GrantedAuthority; import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder; import org.springframework.web.filter.OncePerRequestFilter;
import io.jsonwebtoken.Claims; import io.jsonwebtoken.Jwts;

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);
        }
    }
}
```

JWTAuthorizationFilter

```
else {  
    String jwtToken=request.getHeader(SecurityConstants.HEADER_STRING);  
    if(jwtToken==null || !jwtToken.startsWith(SecurityConstants.TOKEN_PREFIX)) {  
        chain.doFilter(request, response);  return;  
    }  
    Claims claims=Jwts.parser()  
        .setSigningKey(SecurityConstants.SECRET)  
        .parseClaimsJws(jwtToken.replace(SecurityConstants.TOKEN_PREFIX, "")  
        .getBody());  
    String username=claims.getSubject();  
    ArrayList<Map<String, String>> roles=(ArrayList<Map<String, String>>)  
claims.get("roles");  
    Collection<GrantedAuthority> authorities=new ArrayList<>();  
    roles.forEach(r->{  
        authorities.add(new SimpleGrantedAuthority(r.get("authority")));  
    });  
    UsernamePasswordAuthenticationToken authenticationToken=  
new UsernamePasswordAuthenticationToken(username, null,authorities);  
    SecurityContextHolder.getContext().setAuthentication(authenticationToken);  
    chain.doFilter(request, response);  
}  
}}}
```

Application Spring Boot

```
package org.sid;

import org.sid.dao.*; import org.sid.entities.*; import org.sid.service.AccountService;
import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@SpringBootApplication
public class JwtSpringApplication implements CommandLineRunner {
    @Autowired
    private TaskRepository taskRepository;
    @Autowired
    private AccountService accountService;

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

Application Spring Boot

```
@Override  
  
public void run(String... arg0) throws Exception {  
    taskRepository.save(new Task(null, "T1"));  
    taskRepository.save(new Task(null, "T2"));  
    taskRepository.save(new Task(null, "T3"));  
  
    accountService.saveRole(new AppRole(null, "USER"));  
    accountService.saveRole(new AppRole(null, "ADMIN"));  
    accountService.saveUser(new AppUser(null, "user", "1234", null));  
    accountService.saveUser(new AppUser(null, "admin", "1234", null));  
  
    accountService.addRoleToUser("user", "USER");  
    accountService.addRoleToUser("admin", "USER");  
    accountService.addRoleToUser("admin", "ADMIN");  
}  
}
```

Tests : Consulter les listes des taches

Method: GET Request URL: http://localhost:8080/tasks SEND

Parameters ^

Headers

Header name: Content-Type Header value: application/json

Header name: Header value:

ADD HEADER

A ✓

403 Forbidden 1604.81 ms

GET http://localhost:8080/tasks

Response headers (9) Request headers (3)

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
x-frame-options: DENY
content-type: application/json; charset=UTF-8
transfer-encoding: chunked
date: Wed, 07 Feb 2018 11:38:04 GMT

x-content-type-options: nosniff
x-xss-protection: 1; mode=block

{ "timestamp": 1518003484503, "status": 403, "error": "Forbidden", "message": "Access Denied", "path": "/tasks" }

Ajouter un utilisateur

POST ▼ http://localhost:8080/users

Parameters ^

Headers	Body
Body content type application/json	Editor view Raw input

FORMAT JSON MINIFY JSON

```
{"username": "med", "password": "1234", "repassword": "1234"}
```

200 OK 154.08 ms

Copy Download Open Raw

```
{
  "id": 3,
  "username": "med",
  "password": "$2a$10$9AhLVYeaPcjsFYmCS4AhU.w2.CNxMooEVV0DtAo1hZmNgQyPlhveq",
  "roles": [Array[1]
    - 0: {
      "id": 1,
      "role": "USER"
    }
  ],
}
```

Authentification avec le rôle ADMIN

Method POST Request URL http://localhost:8080/login ▼ SEND ⋮

Parameters ^

Headers	Body	Variables
Body content type application/json ▼	Editor view Raw input ▼	
FORMAT JSON MINIFY JSON		
{ "username": "admin", "password": "1234" }		

200 OK 1111.35 ms DETAILS ▾

POST http://localhost:8080/login

Response headers 9 Request headers 2 Redirects 0 Timings

```
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
x-frame-options: DENY
authori Bearer
zation: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbjIiImV4cCI6MTUxODg2ODE2Nywicm9sZXMiolt7ImF1dGhvcmI0eSI6IkFETUl0In0seyJhdXRob3JpdHkiOiJVU0VS
In1dfQ.xc2rophquBp8uBNagMoOURSd9cZklHM6dH6jZJ4E8PDcfKtDfnSb8i17b7TCjWl5a5SReQ4G43Ly7roW-FxZHQ
content-length: 0
date: Wed, 07 Feb 2018 11:49:27 GMT
```

□ ▼ <>

Contenu du JWT

JWT debugger

OPEN JWT FROM ALGORITHM HS256 SHARE THIS JWT

Encoded

```
eyJhbGciOiJIUzUxMiJ9.eyJzd  
WIi0iJhZG1pbIIsImV4cCI6MTU  
xODg2ODE2Nywicm9sZXMiOlt7I  
mF1dGhvcmI0eSI6IkFETUl0In0  
seyJhdXRob3JpdHkiOiJVU0VSI  
n1dfQ.xc2rophquBp8uBNagMo0  
URSd9cZk1HM6dH6jZJ4E8PDcfK  
tDfnSb8il7b7TCjWl5a5SReQ4G  
43Ly7roW-FxZHq
```

Decoded

HEADER:

```
{  
  "alg": "HS512"  
}
```

PAYOUT:

```
{  
  "sub": "admin",  
  "exp": 1518868167,  
  "roles": [  
    {  
      "authority": "ADMIN"  
    },  
    {  
      "authority": "USER"  
    }  
  ]  
}
```

VERIFY SIGNATURE

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

Consulter les tâches

Method Request URL
GET http://localhost:8080/tasks ▼ SEND ⋮

Parameters ^

Headers

<> Toggle source mode + Insert headers set

Header name	Header value
Content-Type	application/json
Header name	Header value
Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbils

ADD HEADER

A

200 OK 346.74 ms

200 OK 32.94 ms

GET http://localhost:8080/tasks

Response headers 9

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
x-frame-options: DENY
content-type: application/json; charset=UTF-8
transfer-encoding: chunked
date: Wed, 07 Feb 2018 15:15:49 GMT

□ ▶ <> ☰

```
[Array[3]
-0: {
  "id": 1,
  "taskName": "T1"
},
-1: {
  "id": 2,
  "taskName": "T2"
},
-2: {
  "id": 3,
  "taskName": "T3"
}]
```

Ajouter une tâche

Method Request URL
POST ▼ http://localhost:8080/tasks SEND

Parameters ^

Headers	Body	Variables
Body content type application/json	Editor view Raw input	
FORMAT JSON MINIFY JSON		
{ "taskName": "JWT" }		
Header name Content-Type	Header value application/json	
Header name Authorization	Header value Bearer eyJhbGciOiJIUzUxMiJ9eyJ	

200 OK 83.11 ms

POST http://localhost:8080/tasks

Response headers 9

```
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
cache-control: no-cache, no-store, max-age=0,
pragma: no-cache
expires: 0
x-frame-options: DENY
content-type: application/json; charset=UTF-8
transfer-encoding: chunked
date: Wed, 07 Feb 2018 15:17:54 GMT
```

{
 "id": 4,
 "taskName": "JWT"
}
med@yousfi.net | ENSET Université Hassan II

Authentification avec le rôle USER

Method POST Request URL http://localhost:8080/login

Parameters ^

Headers

Body content type application/json Editor view Raw input

FORMAT JSON MINIFY JSON

```
{"username":"user","password":"1234"}
```

200 OK 167.36 ms

POST http://localhost:8080/login

Response headers 9

```
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
cache-control: no-cache, no-store, max-age=0
pragma: no-cache
expires: 0
x-frame-options: DENY
authoriza Bearer
tion: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIjB1umFd0mH-2y-eFwNHhGN6VsU254C
content-length: 0
date: Wed, 07 Feb 2018 15:22:45 GMT
```

Authentification avec le rôle USER

The screenshot shows the **JWT debugger** interface. On the left, under **Encoded**, is a large string of characters representing a JWT token. On the right, under **Decoded**, the token is broken down into its components:

- HEADER:**

```
{  
  "alg": "HS512"  
}
```
- PAYLOAD:**

```
{  
  "sub": "user",  
  "exp": 1518880965,  
  "roles": [  
    {  
      "authority": "USER"  
    }  
  ]  
}
```

A red box highlights the **PAYLOAD** section.
- VERIFY SIGNATURE**: A code snippet for HMACSHA256 verification:

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
)  secret base64 encoded
```

Consulter les tâches avec le rôle USER

Method GET Request URL http://localhost:8080/tasks

Parameters ^

Headers

Toggle source mode Insert headers set

Header name Content-Type	Header value application/json
Header name Authorization	Header value Bearer eyJhbGciOiJIUzU

ADD HEADER

A

200 OK 48.87 ms

200 OK 48.87 ms

GET http://localhost:8080/tasks

Response headers 9

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
x-frame-options: DENY
content-type: application/json; charset=UTF-8
transfer-encoding: chunked
date: Wed, 07 Feb 2018 15:27:20 GMT

[
- 0: {
"id": 1,
"taskName": "T1"
},
- 1: {
"id": 2,
"taskName": "T2"
},
- 2: {
"id": 3,
"taskName": "T3"
},
- 3: {
"id": 4,
"taskName": "JWT"
}]

Ajouter une nouvelle tâche avec le rôle USER

Method POST Request URL http://localhost:8080/tasks

Parameters ^

Headers

Body content type application/json Editor view Raw input

FORMAT JSON MINIFY JSON

```
{"taskName": "XXXX"}
```

Header name Header value

Authorization Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwidX^{...}

403 Forbidden 44.82 ms

POST http://localhost:8080/tasks

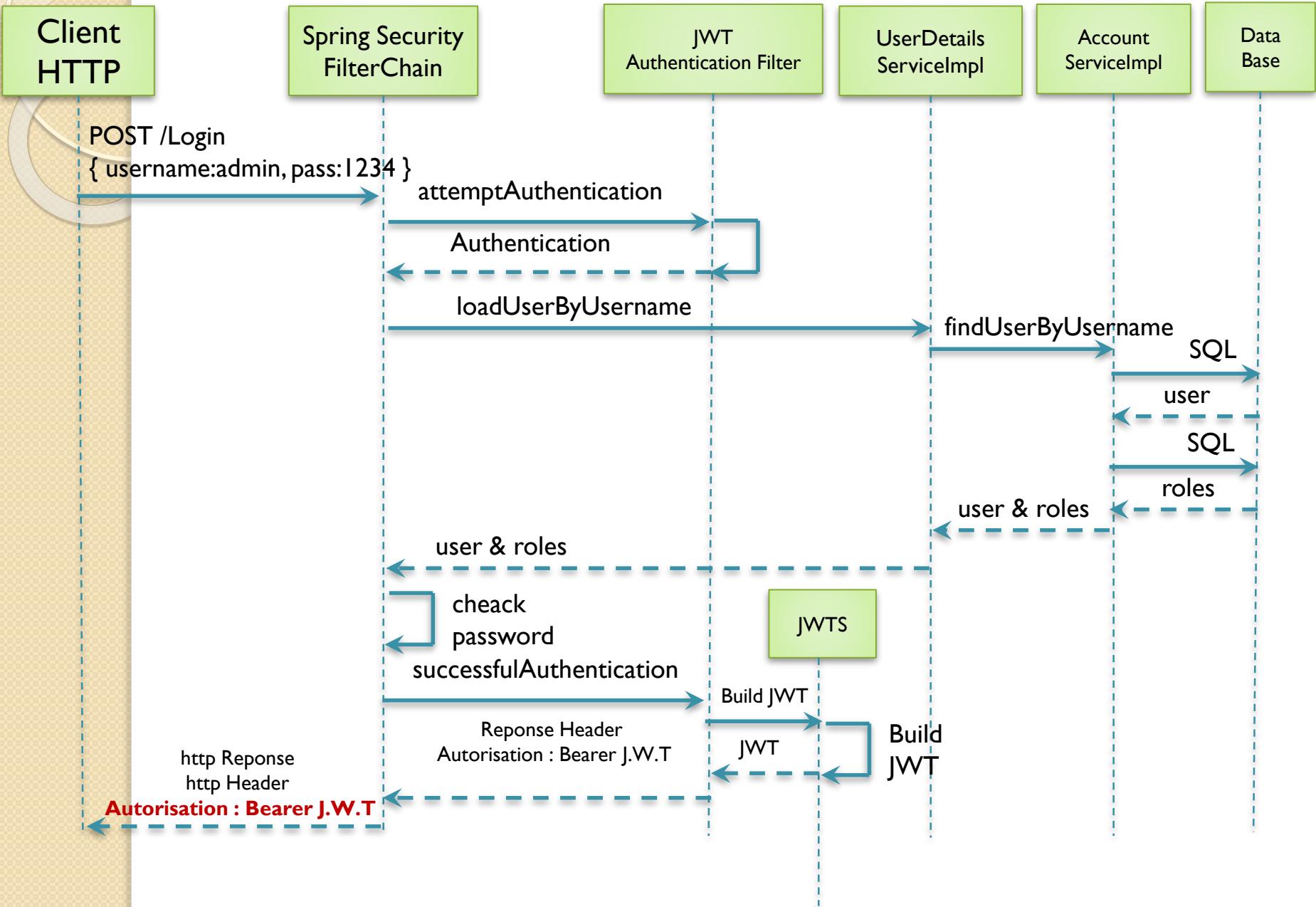
Response headers 9

```
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
x-frame-options: DENY
content-type: application/json; charset=UTF-8
transfer-encoding: chunked
date: Wed, 07 Feb 2018 15:31:10 GMT
```

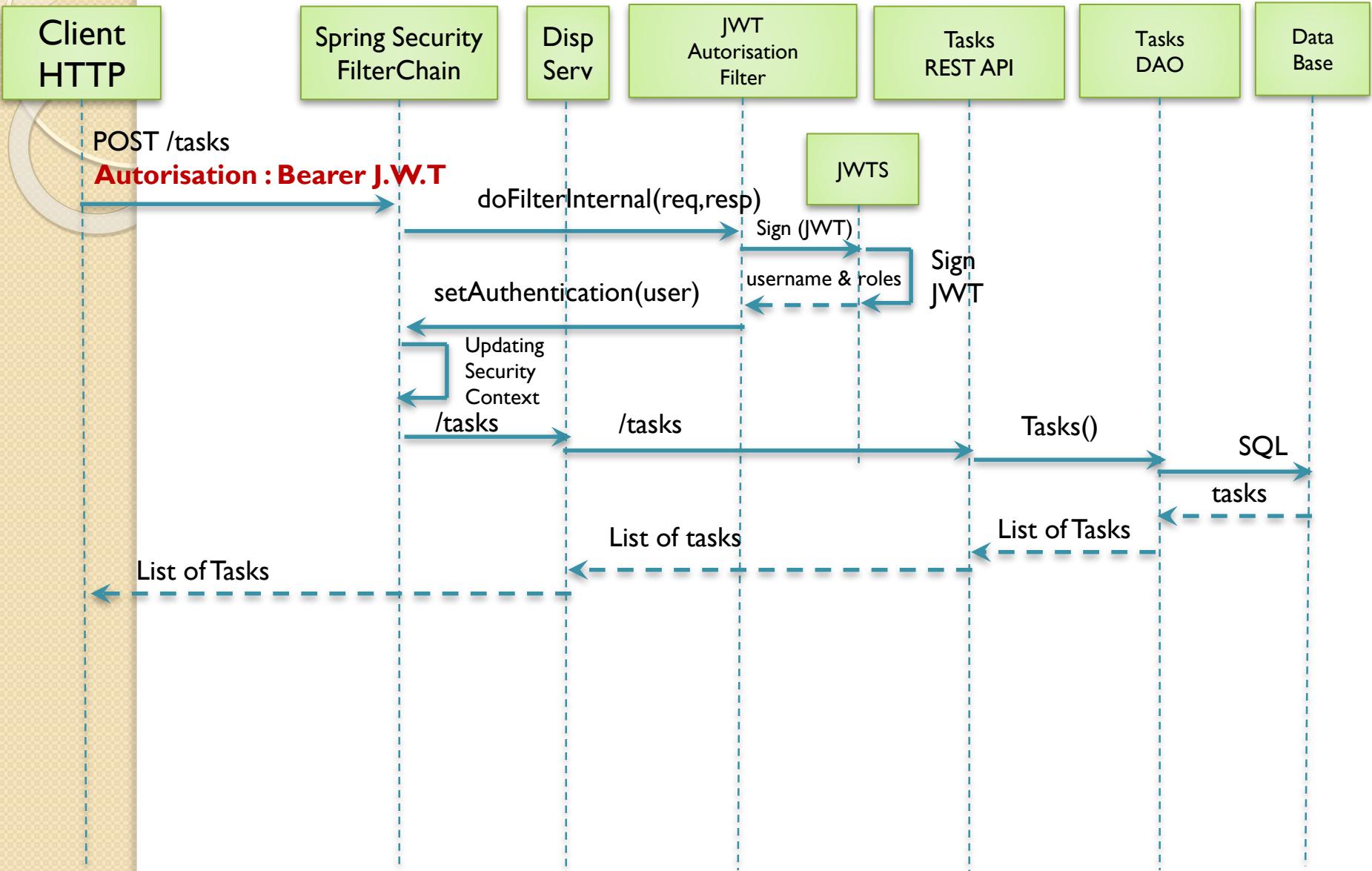
□ □ <> III

```
{
  "timestamp": 1518017470978,
  "status": 403,
  "error": "Forbidden",
  "message": "Access is denied",
  "path": "/tasks"
}
```

Authentification Spring Security avec JWT



Demander une ressource nécessitant l'authentification



Partie FrontEnd Web avec Angular 5

The image shows three browser tabs illustrating the Angular 5 application:

- localhost:4200/login**: The login page features a "Login" button and a "Register New User" button.
- localhost:4200/tasks**: The task list page displays three tasks: T1, T2, and T3.
- localhost:4200/new-task**: The new task creation page shows a "Task Name" input field containing "JWT" and a "Save" button.

The image shows a browser window with the following components:

- localhost:4200/new-task**: Shows a success message: "New task Added" with ID:4 and Task Name:JWT.
- localhost:4200/tasks**: Shows the updated task list with four entries: T1, T2, T3, and JWT.
- Developer Tools (Application tab)**: Shows the browser storage with a token entry:

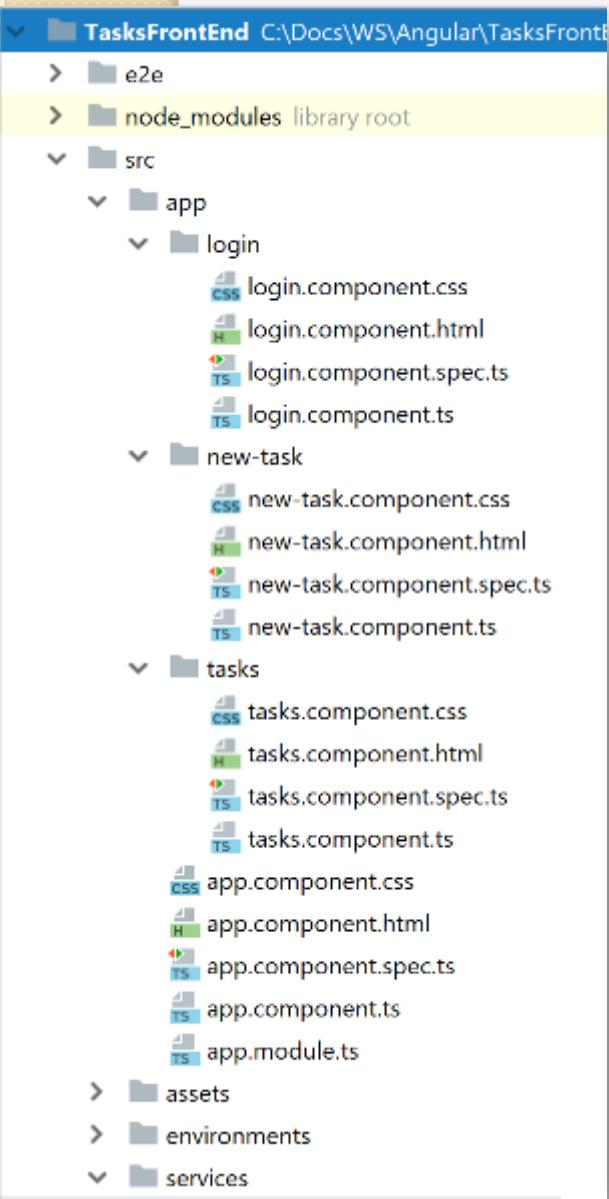
Key	Value
token	Bearer eyJhbGciOiJIUzUxMiJ9eyJzdWliOiJhZG1pbilsIn

Partie FrontEnd Web avec Angular 5

The image displays a screenshot of a web application built with Angular 5, showing multiple browser tabs and windows.

- localhost:4200/login:** Shows an authentication form with "Username" (user) and "Password" (redacted). Buttons for "Login" (red) and "Register New User" (green).
- localhost:4200/tasks:** Shows a table of tasks with columns "ID" and "Task Name". Data: ID 1 (T1), ID 2 (T2), ID 3 (T3), ID 4 (JWT).
- localhost:4200/register:** Shows a registration form with "Username" (hassan), "Password" (redacted), and "Confirm Password" (redacted). A red message box says "You must confirm your password".
- localhost:4200/register:** Shows a registration form with "Username" (mohamed), "Password" (redacted), and "Confirm Password" (redacted). A red message box says "This user already exists, Try with an other username".
- localhost:4200/register:** Shows a registration form with "Username" (mohamed), "Password" (redacted), and "Confirm Password" (redacted). A green message box says "Registration Success". It displays "Your ID :3", "Your username :mohamed", and "Your role:USER". A "Register" button is present.
- localhost:4200/register:** Shows a registration form with "Username" (mohamed), "Password" (redacted), and "Confirm Password" (redacted). A red message box says "This user already exists, Try with an other username".

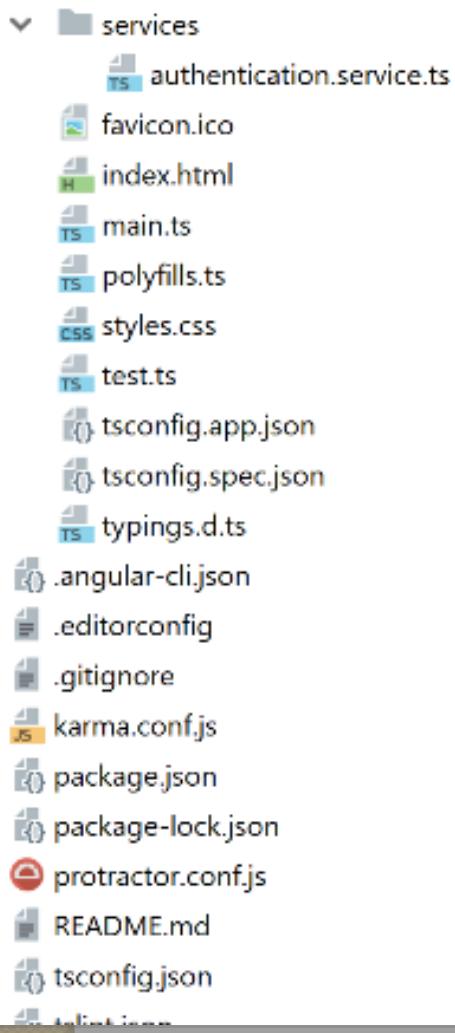
Partie FrontEnd Web avec Angular 5



package.json

```
• "dependencies": {  
    "@angular/animations": "^5.2.0",  
    "@angular/common": "^5.2.0",  
    "@angular/compiler": "^5.2.0",  
    "@angular/core": "^5.2.0",  
    "@angular/forms": "^5.2.0",  
    "@angular/http": "^5.2.0",  
    "@angular/platform-browser": "^5.2.0",  
    "@angular/platform-browser-dynamic": "^5.2.0",  
    "@angular/router": "^5.2.0",  
    "angular2-jwt": "^0.2.3",  
    "bootstrap": "^3.3.7",  
    "core-js": "^2.4.1",  
    "rxjs": "^5.5.6",  
    "zone.js": "^0.8.19"  
},
```

Partie FrontEnd Web avec Angular 5



package.json

```
"devDependencies": {
    "@angular/cli": "1.6.8",
    "@angular/compiler-cli": "^5.2.0",
    "@angular/language-service": "^5.2.0",
    "@types/jasmine": "~2.8.3",
    "@types/jasminewd2": "~2.0.2",
    "@types/node": "~6.0.60",
    "codelyzer": "^4.0.1",
    "jasmine-core": "~2.8.0",
    "jasmine-spec-reporter": "~4.2.1",
    "karma": "~2.0.0",
    "karma-chrome-launcher": "~2.2.0",
    "karma-coverage-istanbul-reporter": "^1.2.1",
    "karma-jasmine": "~1.1.0",
    "karma-jasmine-html-reporter": "^0.2.2",
    "protractor": "~5.1.2",
    "ts-node": "~4.1.0",
    "tslint": "~5.9.1",
    "typescript": "~2.5.3"
}
```

.angular-cli.json

```
"styles": [  
  "styles.css",  
  "./node_modules/bootstrap/dist/css/bootstrap.css"  
],  
"scripts": [] ,
```

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import { TasksComponent } from './tasks/tasks.component';
import { NewTaskComponent } from './new-task/new-task.component';
import { RouterModule, Routes} from '@angular/router';
import {FormsModule} from '@angular/forms';
import {HttpClientModule} from '@angular/common/http';
import {AuthenticationService} from '../services/authentication.service';
import { RegisterComponent } from './register/register.component';

const appRoutes:Routes=[  
  {path:"login",component:LoginComponent},  
  {path:"tasks",component:TasksComponent},  
  {path:"new-task",component>NewTaskComponent},  
  {path:"register",component:RegisterComponent},  
  {path:'',redirectTo:'/login',pathMatch:'full'}  
];  
@NgModule({  
  declarations: [  
    AppComponent,  
    LoginComponent,  
    TasksComponent,  
    NewTaskComponent,  
    RegisterComponent  
  ],  
  imports: [  
    BrowserModule, FormsModule, RouterModule.forRoot(appRoutes), HttpClientModule  
  ],  
  providers: [AuthenticationService],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```

authentication.service.ts

```
import {Injectable} from '@angular/core';
import "rxjs/Rx"
import {HttpClient, HttpHeaders} from '@angular/common/http';
import {JwtHelper} from 'angular2-jwt';
@Injectable()
export class AuthenticationService{
  private host:string="http://localhost:8080";
  private jwtToken:string;
  private roles:Array<any>=[];
  constructor(private http:HttpClient) {}

  login(user) {
    return this.http.post(this.host+"/login",user,{ observe: 'response' })
  }

  register(user) {
    return this.http.post(this.host+"/users",user);
  }

  saveToken(jwtToken) {
    this.jwtToken=jwtToken;
    localStorage.setItem("token",jwtToken);
    let jwtHelper=new JwtHelper();
    this.roles=jwtHelper.decodeToken(this.jwtToken).roles;
  }
}
```

authentication.service.ts

```
getTasks() {
    if(this.jwtToken==null) this.loadToken();
    return this.http.get(this.host+"/tasks", {headers:new HttpHeaders({ 'authorization':this.jwtToken}) });
}
saveTask(task) {
    let headers=new HttpHeaders();
    headers.append('authorization',this.jwtToken);
    return this.http.post(this.host+"/tasks",task, {headers:new HttpHeaders({ 'authorization':this.jwtToken}) });
}
loadToken() {
    this.jwtToken=localStorage.getItem('token');
    return this.jwtToken;
}
logout() {
    localStorage.removeItem('token');
}
isAdmin() {
    for(let r of this.roles) {
        if(r.authority=='ADMIN') return true;
    }
    return false;
}
```

app.component.html

```
<p></p>
<div class="container">
  <button class="btn btn-default"
(click)="onLogout()">Logout</button>
  <button class="btn btn-default" routerLink="/tasks">Tasks</button>
</div>
<p></p>
<div class="container">
  <router-outlet></router-outlet>
</div>
```

app.component.ts

```
import { Component } from '@angular/core';
import { AuthenticationService } from '../services/authentication.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  constructor(private authService: AuthenticationService, private router: Router) {}

  onLogout() {
    this.authService.logout();
    this.router.navigateByUrl('/login');
  }
}
```

login.component.html

```
<p></p>
<div class="container">
  <div class="alert alert-danger" *ngIf="mode==1">
    <strong>Bad credentials!</strong> Invalid username or password.
  </div>
  <div class="row col-md-6 col-md-offset-3">
    <div class="panel panel-primary">
      <div class="panel-heading">Authentification</div>
      <div class="panel-body">
        <form #f="ngForm" (ngSubmit)="onLogin(f.value)">
          <div class="form-group">
            <label class="control-label">Username</label>
            <input type="text" name="username" ngModel="" class="form-control" required />
          </div>
          <div class="form-group">
            <label class="control-label">Password</label>
            <input type="password" name="password" ngModel="" class="form-control" required />
          </div>
          <button type="submit" class="btn btn-danger">Login</button>
          <button (click)="onRegister()" class="btn btn-success" [disabled]="!f.valid">Register New User</button>
        </form>
        <p></p>
      </div>
    </div>
  </div>
</div>
```

login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Http } from "@angular/http";
import "rxjs/add/operator/map";
import { AuthenticationService } from "../../services/authentication.service";
import { Router } from "@angular/router";
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  private mode=0;

  constructor(private authService:AuthenticationService,
    private router:Router) {}

  ngOnInit() {
    let token=this.authService.loadToken();
    if(token)
      this.router.navigateByUrl("/tasks");
  }
  onLogin(formData) {
    this.authService.login(formData)
      .subscribe(resp=>{
        let jwtToken=resp.headers.get('authorization');
        this.authService.saveToken(jwtToken);
        this.router.navigateByUrl("/tasks");
      },
      err=>{
        this.mode=1;
      })
  }
  onRegister() {
    this.router.navigateByUrl("/register");
  }
}
```

tasks.component.html

```
<div class="container">
  <button *ngIf="authService.isAdmin()" class="btn btn-danger"
    (click)="onNewTask()">New Task</button>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>ID</th><th>Task Name</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let t of tasks">
        <td>{{ t.id }}</td>
        <td>{{ t.taskName }}</td>
      </tr>
    </tbody>
  </table>
</div>
```

tasks.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AuthenticationService } from
"../../services/authentication.service";
import { Router } from "@angular/router";

@Component({
  selector: 'app-tasks',
  templateUrl: './tasks.component.html',
  styleUrls: ['./tasks.component.css']
})
export class TasksComponent implements OnInit {
  public tasks:any;
  constructor(public authService:AuthenticationService,
             private router:Router) { }
  ngOnInit() {

    this.authService.getTasks()
      .subscribe(data=>{
        this.tasks=data;
      },err=>{
        this.authService.logout();
        this.router.navigateByUrl("/login");
      })
  }
  onNewTask() {
    this.router.navigateByUrl('/new-task');
  }
}
```

new-task.component.html

```
<p></p>
<div class="container">
  <div class="alert alert-danger" *ngIf="mode==0 && errorMessage">
    <strong>{{errorMessage}}</strong>
  </div>
  <div class="panel panel-success" *ngIf="mode==1">
    <div class="panel-heading">
      <strong>Registration Success</strong>
    </div>
    <div class="panel-body">
      <ul class="list-group">
        <li class="list-group-item">
          Your ID :{{user.id}}
        </li>
        <li class="list-group-item">
          Your username :{{user.username}}
        </li>
        <li class="list-group-item">
          Your role:{{user.roles[0].role}}
        </li>
      </ul>
    </div>
  </div>
</div>
```

new-task.component.html

```
<div class="row col-md-6 col-md-offset-3" *ngIf="mode==0">
  <div class="panel panel-primary">
    <div class="panel-heading">Registration</div>
    <div class="panel-body">
      <form #f="ngForm" (ngSubmit)="onRegister(f.value)">
        <div class="form-group">
          <label class="control-label">Username</label>
          <input type="text" name="username" ngModel="" class="form-control"/>
        </div>
        <div class="form-group">
          <label class="control-label">Password</label>
          <input type="password" name="password" ngModel="" class="form-control"/>
        </div>
        <div class="form-group">
          <label class="control-label">Confirm Password</label>
          <input type="password" name="repassword" ngModel="" class="form-control"/>
        </div>
        <button type="submit" class="btn btn-danger">Register</button>
      </form>
    </div>
  </div>
</div>
```

new-task.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AuthenticationService } from '../../services/authentication.service';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
  user:any;
  mode:number=0;
  errorMessage:string;
  constructor(private authService:AuthenticationService) { }

  ngOnInit() {
  }
  onRegister(user) {
    this.authService.register(user)
      .subscribe(data=>{
        this.user=data;
        this.mode=1;
      },
      err=>{
        this.errorMessage=err.error.message;
        this.mode=0;
      })
  }
}
```

register.component.html

```
<p></p>
<div class="container">
  <div class="alert alert-danger" *ngIf="mode==0 && errorMessage">
    <strong>{{errorMessage}}</strong>
  </div>
  <div class="panel panel-success" *ngIf="mode==1">
    <div class="panel-heading">
      <strong>Registration Success</strong>
    </div>
    <div class="panel-body">
      <ul class="list-group">
        <li class="list-group-item">
          Your ID :{{user.id}}
        </li>
        <li class="list-group-item">
          Your username :{{user.username}}
        </li>
        <li class="list-group-item">
          Your role:{{user.roles[0].role}}
        </li>
      </ul>
    </div>
  </div>
</div>
```

register.component.html

```
<div class="row col-md-6 col-md-offset-3" *ngIf="mode==0">
  <div class="panel panel-primary">
    <div class="panel-heading">Registration</div>
    <div class="panel-body">
      <form #f="ngForm" (ngSubmit)="onRegister(f.value)">
        <div class="form-group">
          <label class="control-label">Username</label>
          <input type="text" name="username" ngModel="" class="form-control"
required />
        </div>
        <div class="form-group">
          <label class="control-label">Password</label>
          <input type="password" name="password" ngModel="" class="form-control"
required />
        </div>
        <div class="form-group">
          <label class="control-label">Confirm Password</label>
          <input type="password" name="repassword" ngModel="" class="form-control"
required />
        </div>
        <button type="submit" class="btn btn-danger" [disabled]={!f.valid}>
Register</button>
      </form>
    </div>
  </div>
</div>
```

register.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AuthenticationService } from '../../services/authentication.service';

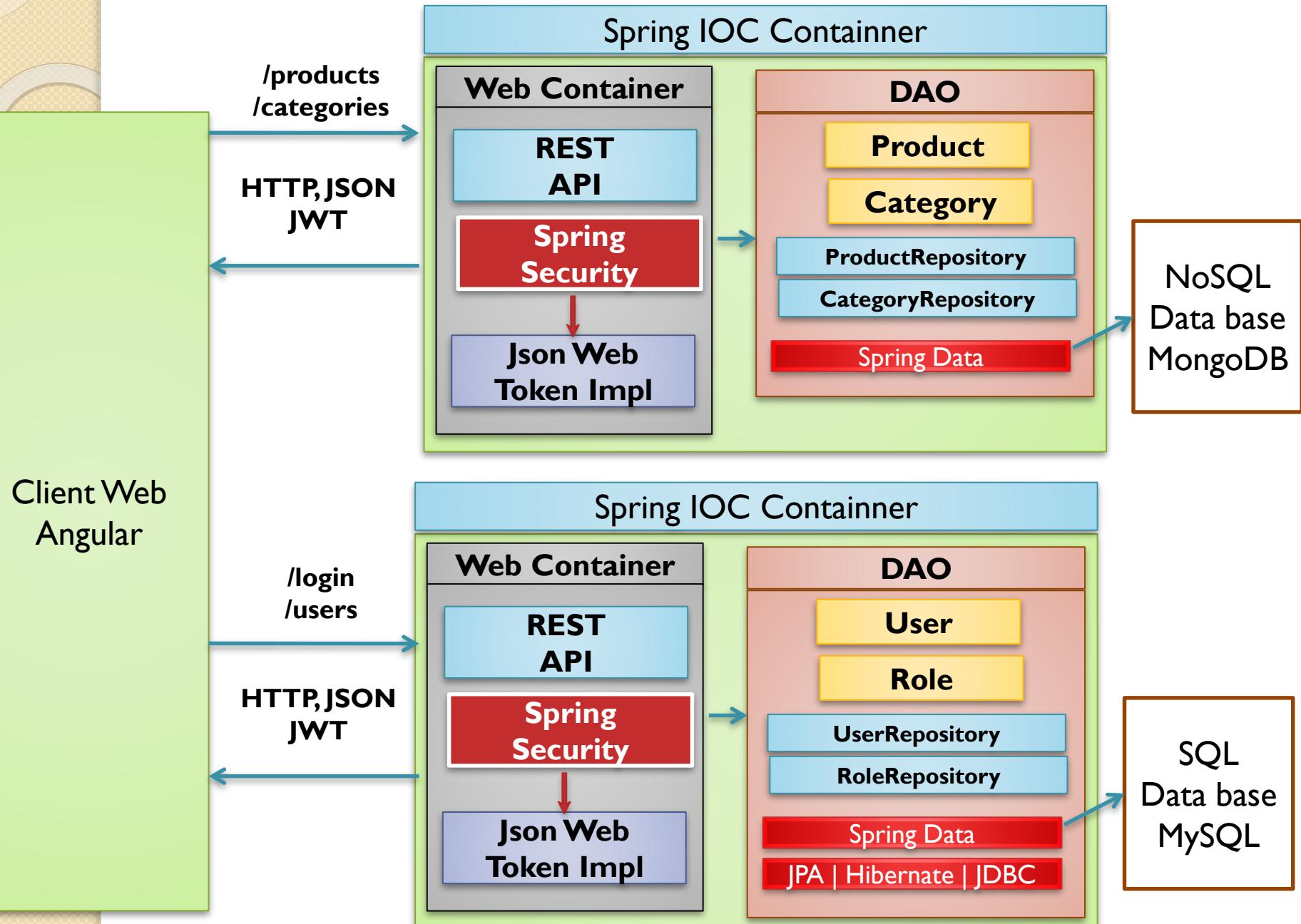
@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
  user:any;
  mode:number=0;
  errorMessage:string;
  constructor(private authService:AuthenticationService) { }

  ngOnInit() {
  }
  onRegister(user) {
    this.authService.register(user)
      .subscribe(data=>{
        this.user=data;
        this.mode=1;
      },
      err=>{
        this.errorMessage=err.error.message;
        this.mode=0;
      })
  }
}
```

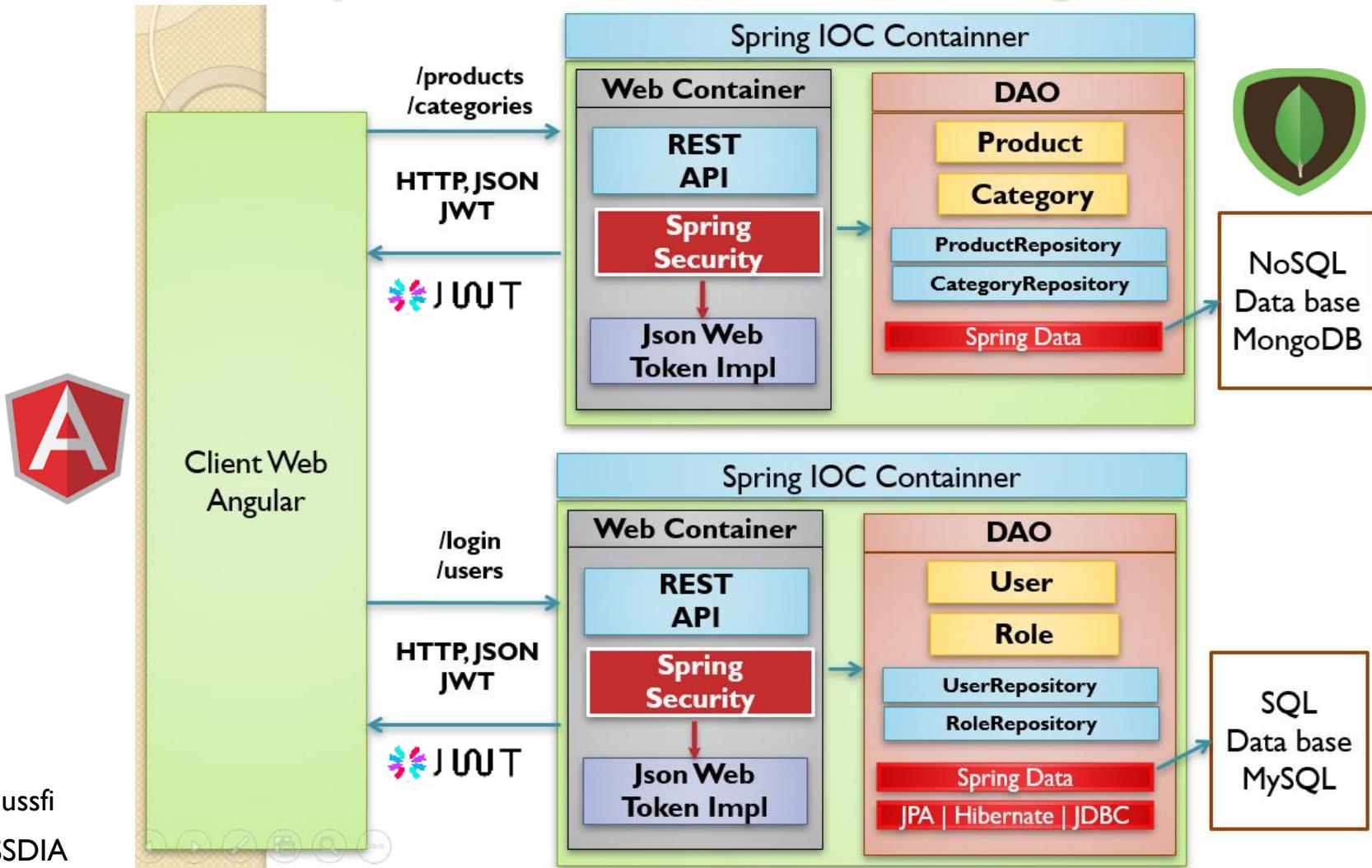
Application de synthèse Micro services, MongoDB et sécurité avec Json Web Token (JWT)

- Créer une application qui permet de gérer des produits appartenant chacun à une catégorie. Une catégorie contient plusieurs produits.
- L'application se compose de deux micro services:
 - Un micro service qui permet de gérer les produits et les catégories.
 - Les données sont stockées dans une base de données MongoDB
 - L'accès à ce micro service d'avoir une autorisation. L'utilisateur ayant le rôle ADMIN peut gérer les produits et les catégories et l'utilisateur ayant le rôle USER peut gérer uniquement les produits.
 - Un Micro service qui se charge de centraliser l'opération d'authentification et de gestion des autorisations en utilisant Json Web Token. Dans ce miro-service, on suppose que les utilisateurs et les rôles sont stockés dans une base de données MySQL.
- Travail à faire :
 - Créer le micro-service de gestion des produits et des catégories
 - Créer le micro service de gestion de l'authentification et des autorisations

Architecture



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



Mohamed Youssfi
Laboratoire SSDIA

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

