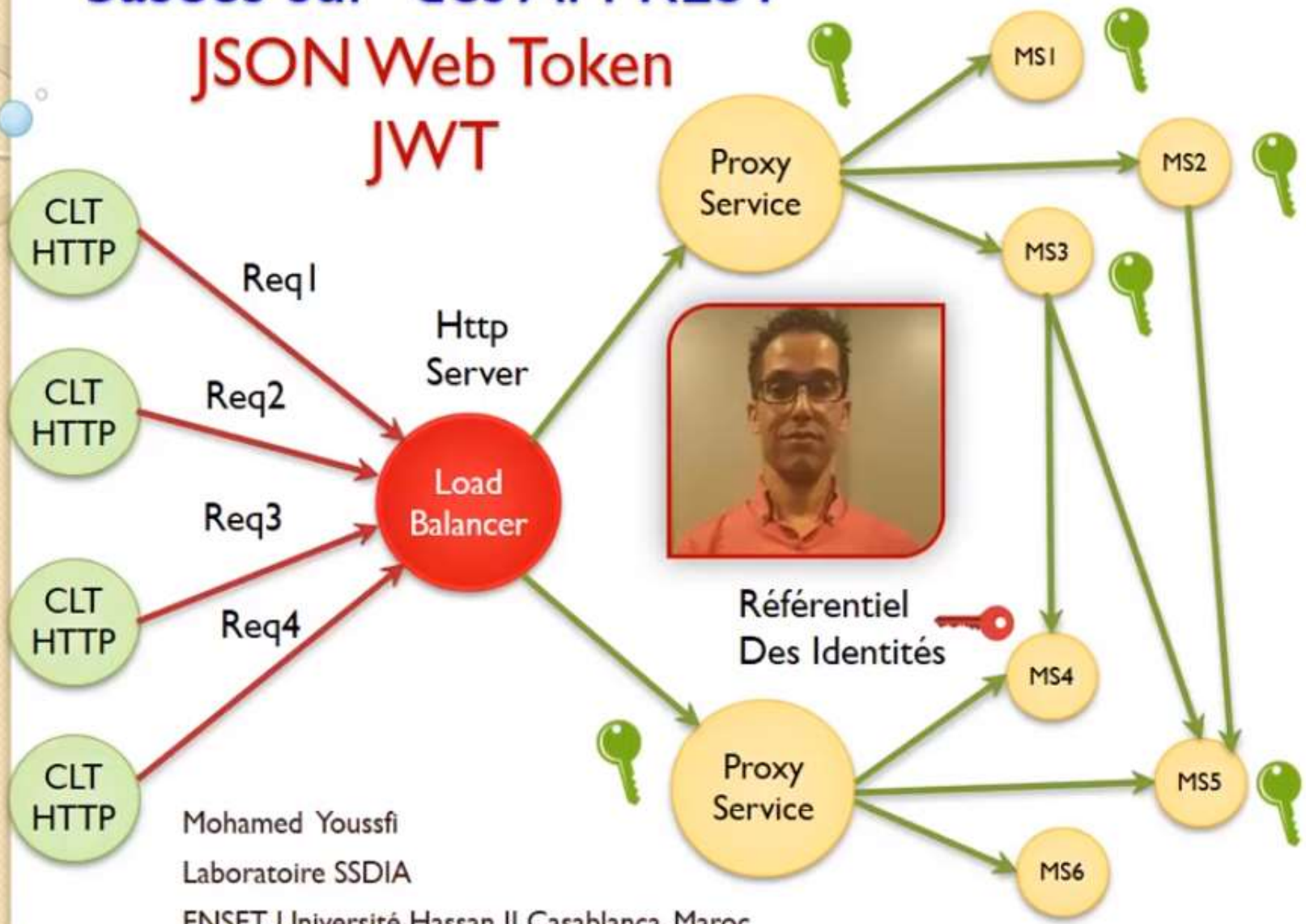
Sécurité des applications web basées sur des API REST

JSON Web Token

JWT



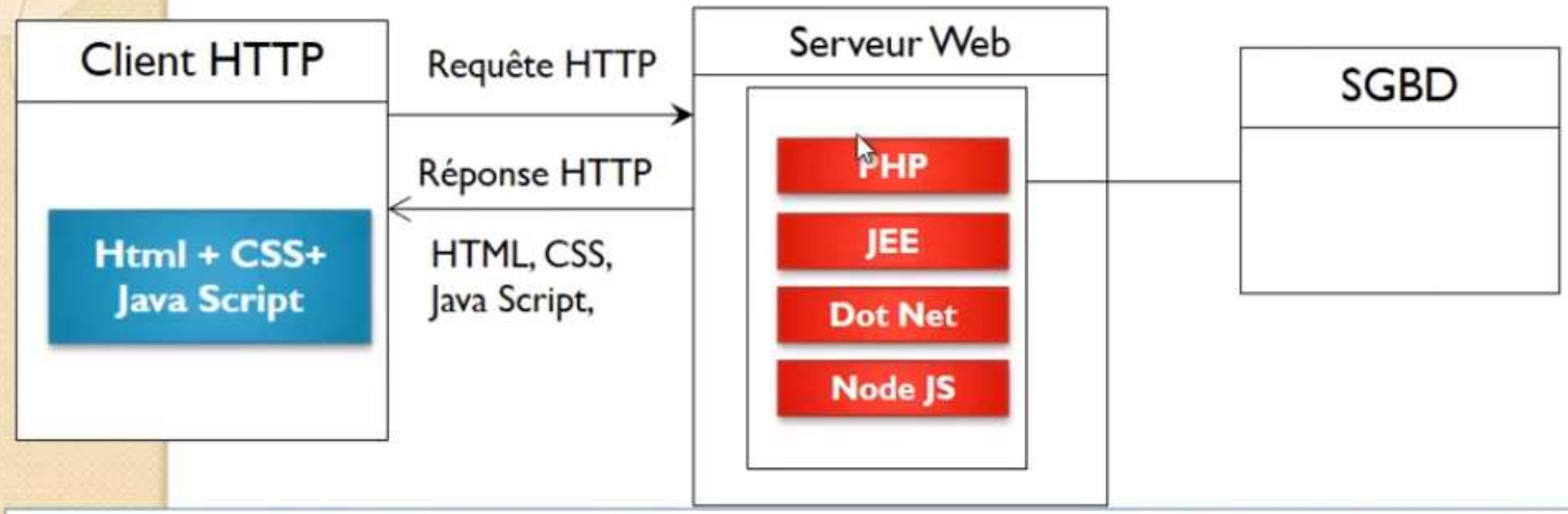
Mohamed Youssfi

Laboratoire SSDIA

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

Architecture Web



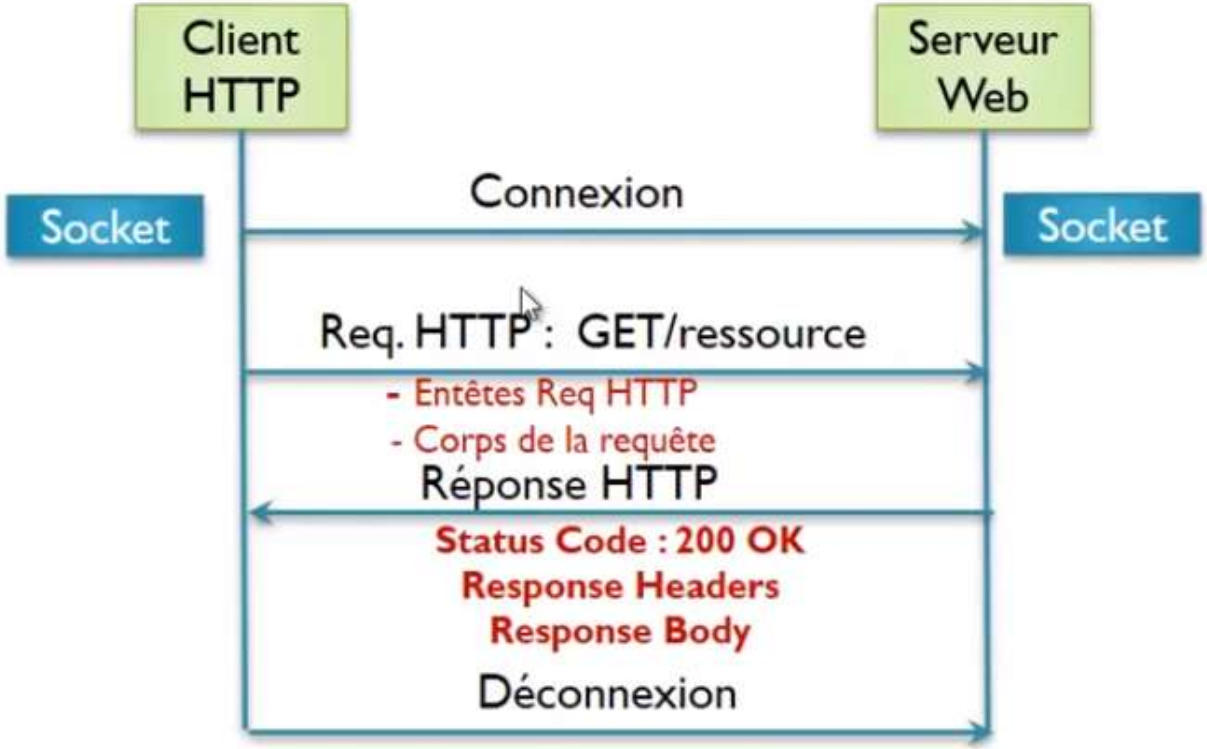
- Un client web (Browser) communique avec le serveur web (Apache) en utilisant le protocole HTTP
- Une application web se compose de deux parties:
  - La partie Backend : S'occupe des traitements effectués coté serveur :
    - Technologies utilisées : PHP, JEE, .Net, Node JS
  - La partie Front end : 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

LE PROTOCOLE HTTP

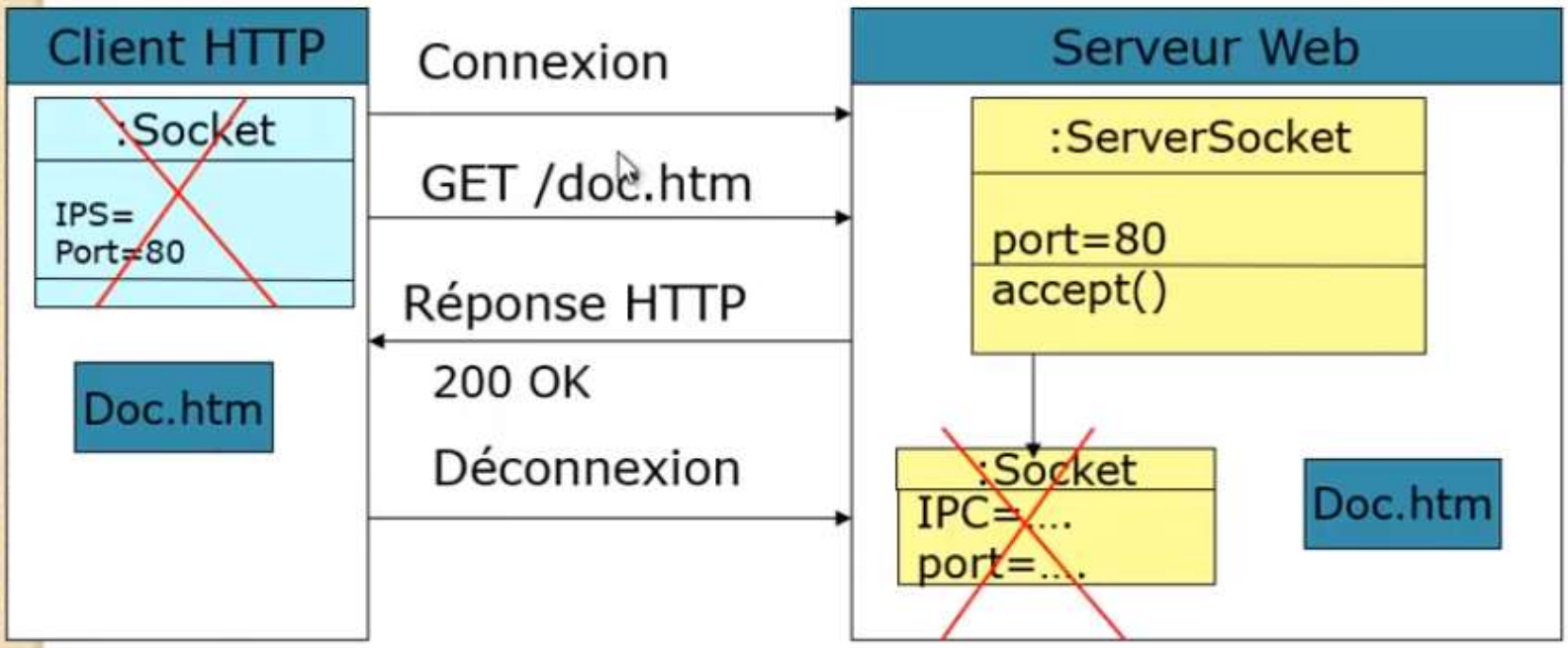
- **HTTP :HyperText Tranfert Protocol**
  - Protocole qui permet au client de récupérer des documents du serveur
  - Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP…)
  - Ce protocole permet également de soumissionner les formulaires
- **Fonctionnement (très simple en HTTP/1.0)**
  - Le client se connecte au serveur (Créer une socket)
  - Le client demande au serveur un document : Requête HTTP
  - Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
  - Déconnexion

LE PROTOCOLE HTTP

- **Fonctionnement (très simple en HTTP/1.0)**
  - Le client se connecte au serveur (Créer une socket)
  - Le client demande au serveur un document : Requête HTTP
  - Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
  - Déconnexion



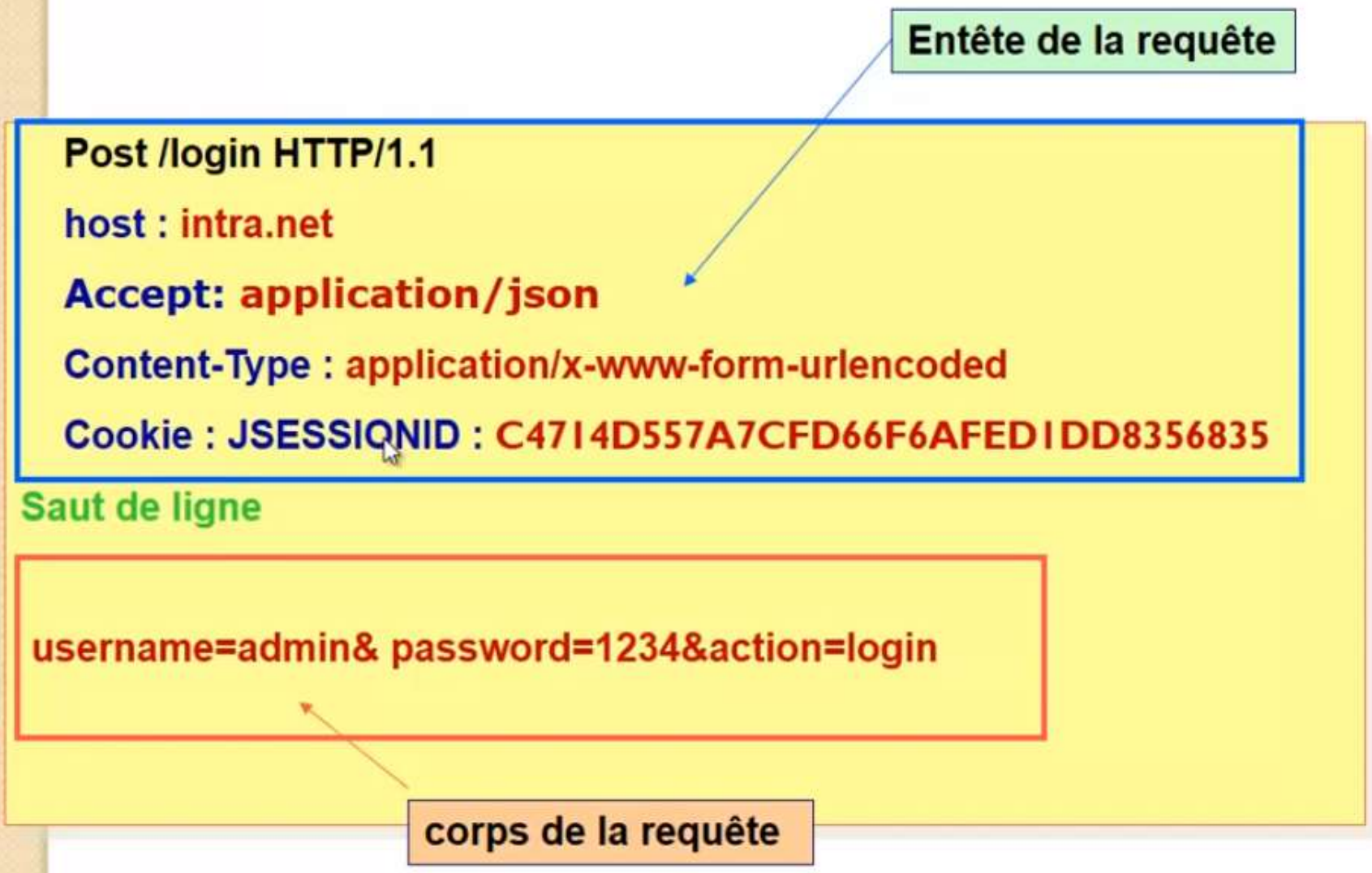
Connexion


Client HTTP
:Socket
IPS=
Port=80
Doc.htm
Connexion
GET /doc.htm
Réponse HTTP
200 OK
Déconnexion
Serveur Web
:ServerSocket
port=80
accept()
:Socket
IPC=...
port=....
Doc.htm

Utilisation des méthodes HTTP dans le cas d'une API REST

- Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:
  - GET : Pour Consulter des ressources
  - POST : Pour soumissionner et ajouter de nouvelles ressources
  - PUT pour mettre à jour des ressources existantes
  - DELETE pour supprimer des ressources existantes.
  - OPTIONS : Permet au client HTTP d'intéroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisés, etc…). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).
  - HEAD permet de consulter les métadonnées d'une ressource (Type, Capacité, Date de dernière modification etc…)

Exemple de requête HTTP avec POST



Exemple de requête HTTP avec POST

Entête de la requête

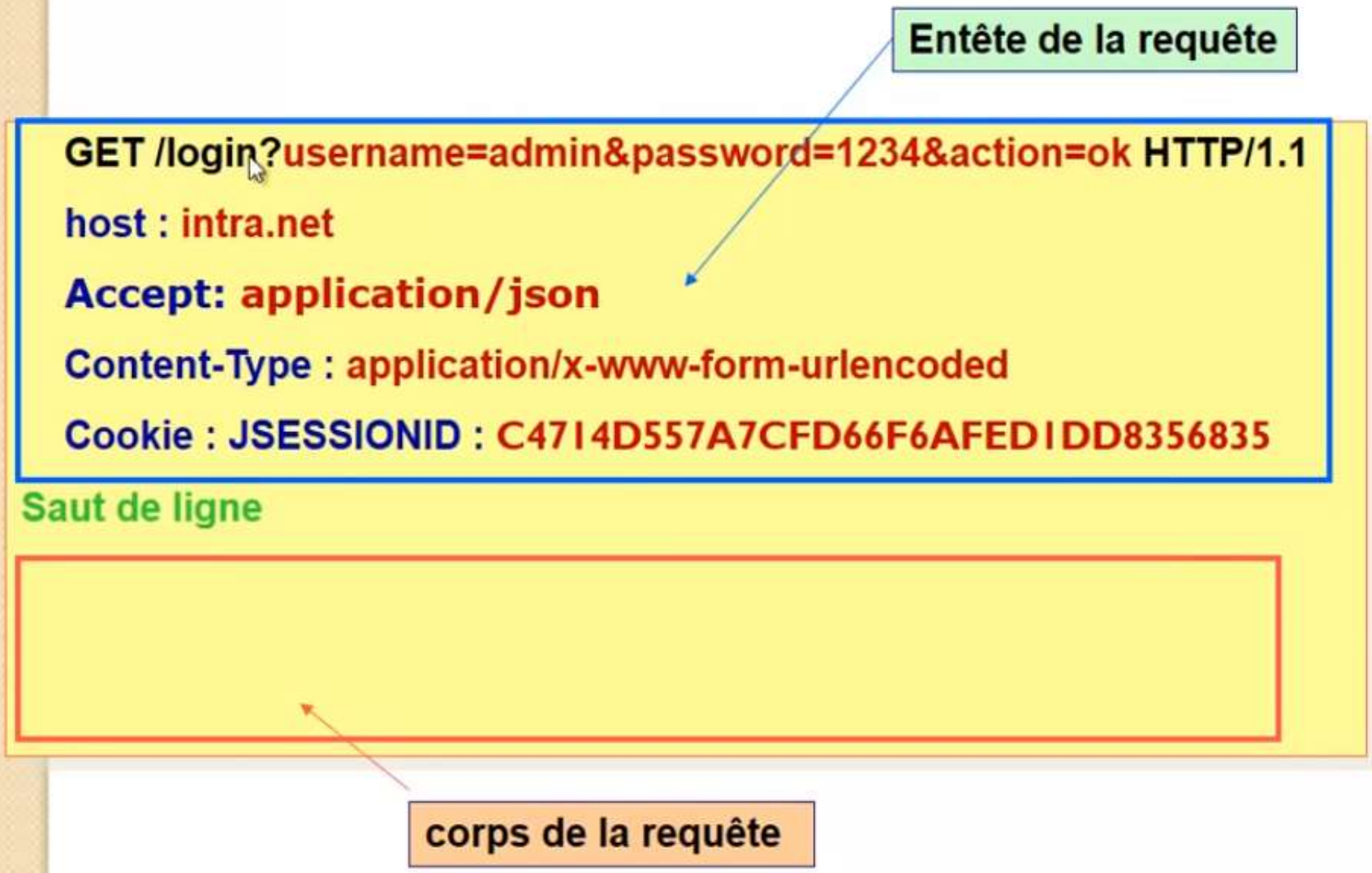
```
Post /login HTTP/1.1
host : intra.net
Accept: application/json
Content-Type : application/json
Cookie : JSESSIONID : C4714D557A7CFD66F6AFED1DD8356835
```

Saut de ligne

```
{"username":"admin","password":"1234","action":"login" }
```

corps de la requête

Exemple de requête HTTP avec GET



Réponse HTTP:

Entête de la réponse

HTTP/1.1 200 OK

Date : Wed, 05Feb02 15:02:01 GMT

Server : Apache/1.3.24

Last-Modified : Wed 02Oct01 24:05:01GMT

Content-Type : application/json

Content-legnth : 77

Set-Cookie : JSESSIONID: C4714D557A7CFD66F6AFED1DD8356835

Saut de ligne

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

corps de la réponse

Code de status

- Lorsque le serveur renvoie un document, il lui associe un code de statut renseignant ainsi le client sur le résultat de la requête (requête invalide, document non trouvé…).
- Les principales valeurs des codes de statut HTTP sont détaillées dans le tableau ci-après.
- Information 1xx :
  - 100 (Continue) : Utiliser dans le cas où la requête possède un corps.
  - 101 (Switching protocol) : Demander au client de changer de protocole. Par exemple de Http1.0 vers Http 1.1
- Succès 2xx :
  - 200 (OK) : Le document a été trouvé et son contenu suit
  - 201 (Created) : Le document a été créé en réponse à un PUT
  - 202 (Accepted) : Requête acceptée, mais traitement non terminé
  - 204 (No response) : Le serveur n'a aucune information à renvoyer
  - 206 (Partial content) : Une partie du document suit

Code de status

- Redirection 3xx :
  - 301 (Moved) : Le document a changé d'adresse de façon permanente
  - 302 (Found) : Le document a changé d'adresse temporairement
  - 304 (Not modified) : Le document demandé n'a pas été modifié
- Erreurs du client 4xx :
  - 400 (Bad request) : La syntaxe de la requête est incorrecte
  - 401 (Unauthorized) : Le client n'a pas les privilèges d'accès au document
  - 403 (Forbidden) : L'accès au document est interdit
  - 404 (Not found) : Le document demandé n'a pu être trouvé
  - 405 (Method not allowed): La méthode de la requête n'est pas autorisée
- Erreurs du serveur 5xx :
  - 500 (Internal error) : Une erreur inattendue est survenue au niveau du serveur
  - 501 (Not implemented) : La méthode utilisée n'est pas implémentée
  - 502 (Bad gateway) : Erreur de serveur distant lors d'une requête proxy

Entêtes HTTP

- Entêtes HTTP génériques :
  - Content-length : Longueur en octets des données suivant les en-têtes
  - Content-type : Type MIME des données qui suivent
  - Connection : Indique si la connexion TCP doit rester ouverte (Keep-Alive) ou être fermée (close)
- Entêtes de la requête :
  - Accept : Types MIME que le client accepte
  - Accept-encoding: Méthodes de compression supportées par le client
  - Accept-language: Langues préférées par le client (pondérées)
  - Cookie : Données de cookie mémorisées par le client
  - Host : Hôte virtuel demandé
  - If-modified-since : Ne retourne le document que si modifié depuis la date indiquée
  - If-none-match : Ne retourne le document que s'il a changé
  - Referer : URL de la page à partir de laquelle le document est demandé
  - User-agent : Nom et version du logiciel client

Entêtes de réponse

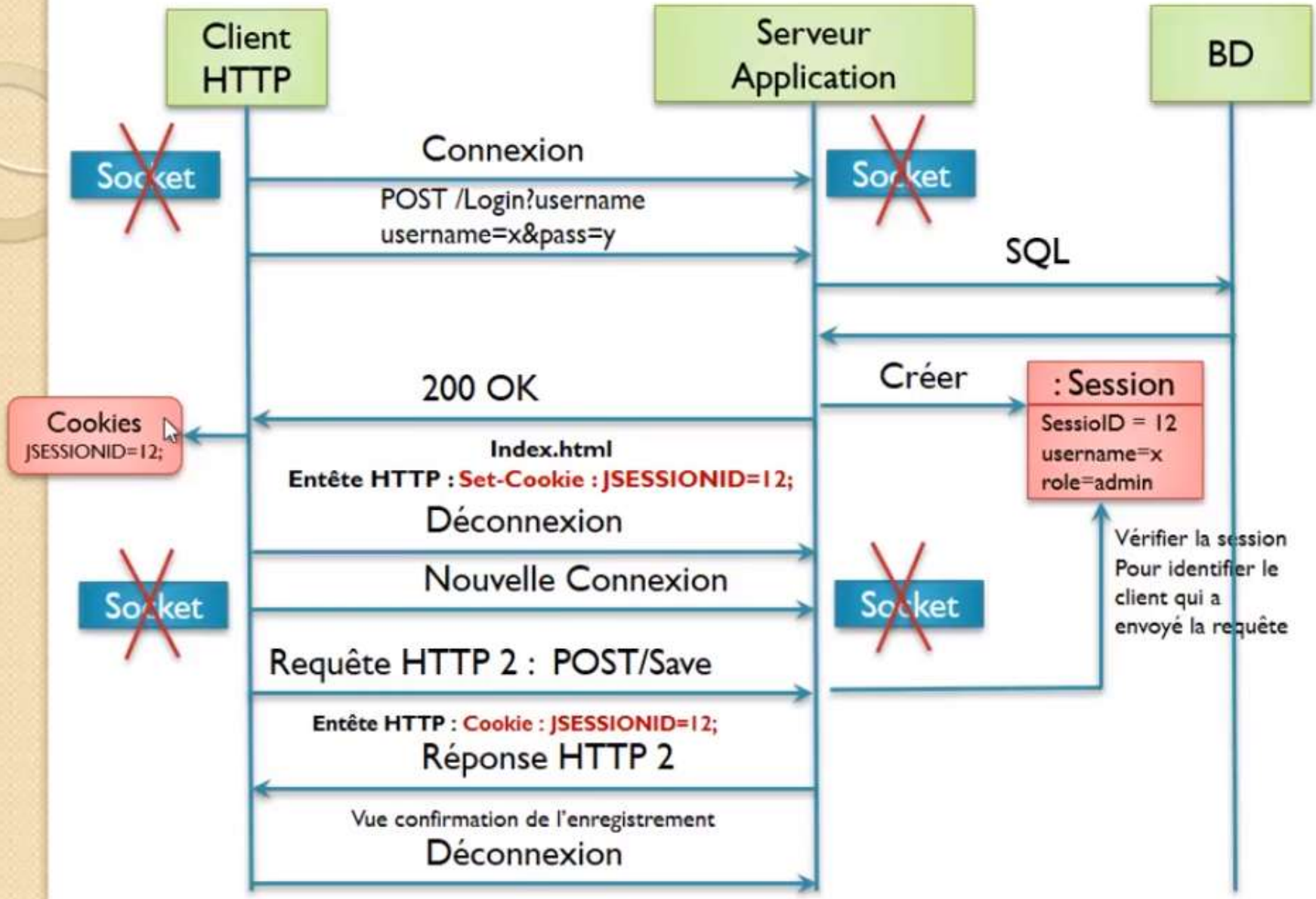
- Allowed : Méthodes HTTP autorisées pour cette URI (comme POST)
- Content-encoding : Méthode de compression des données qui suivent
- Content-language : Langue dans laquelle le document retourné est écrit
- Date : Date et heure UTC courante
- Expires : Date à laquelle le document expire
- Last-modified : Date de dernière modification du document
- Location : Adresse du document lors d'une redirection
- Etag : Numéro de version du document
- Pragma : Données annexes pour le navigateur (par exemple, no.cache)
- Server : Nom et version du logiciel serveur
- Set-cookie : Permet au serveur d'écrire un cookie sur le disque du client

Authentification web basée sur

Les sessions et Cookies

med@youssfi.net

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.

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.

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 provoquants 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.

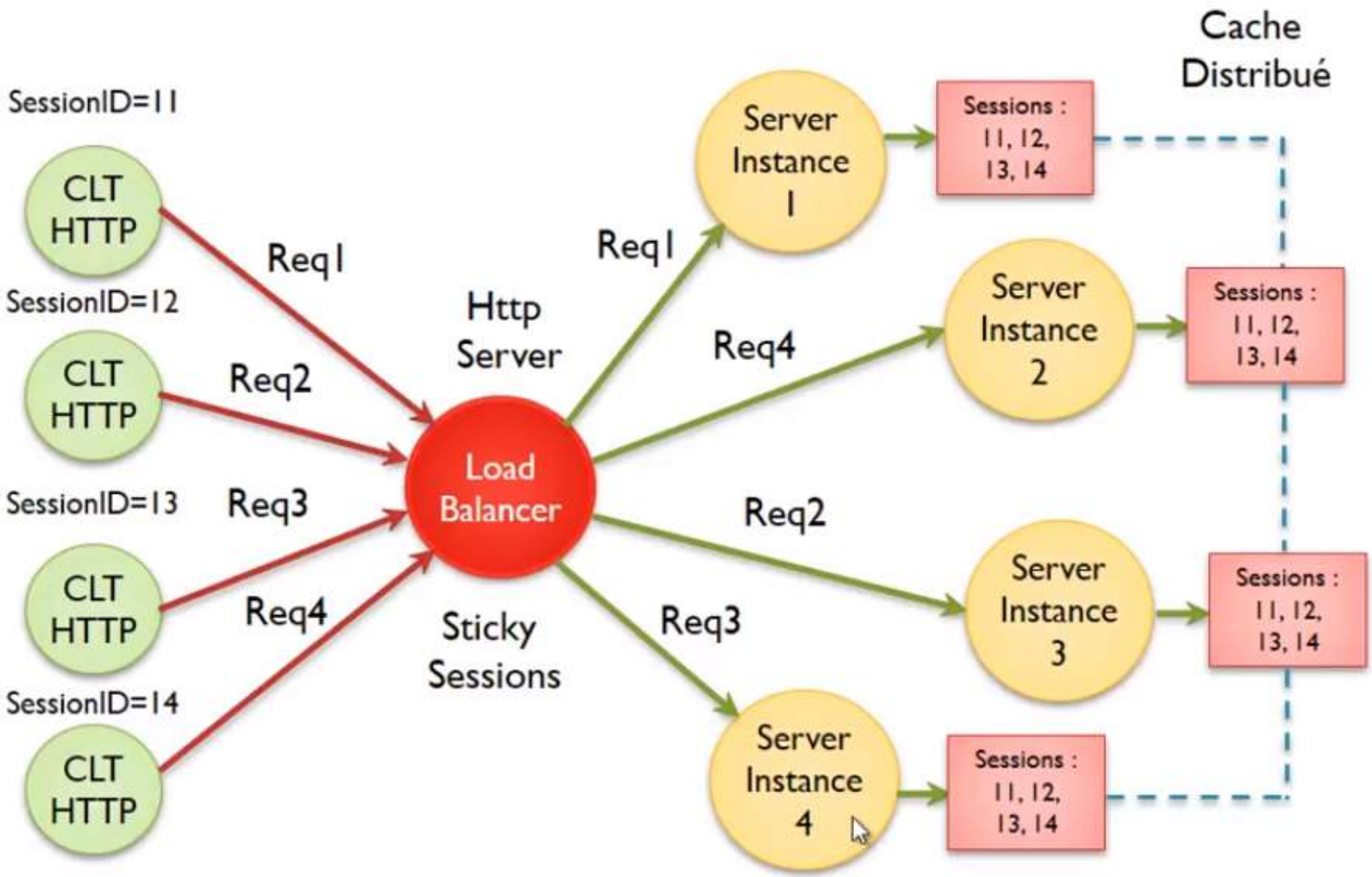
Parades de XSS

- 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 ;

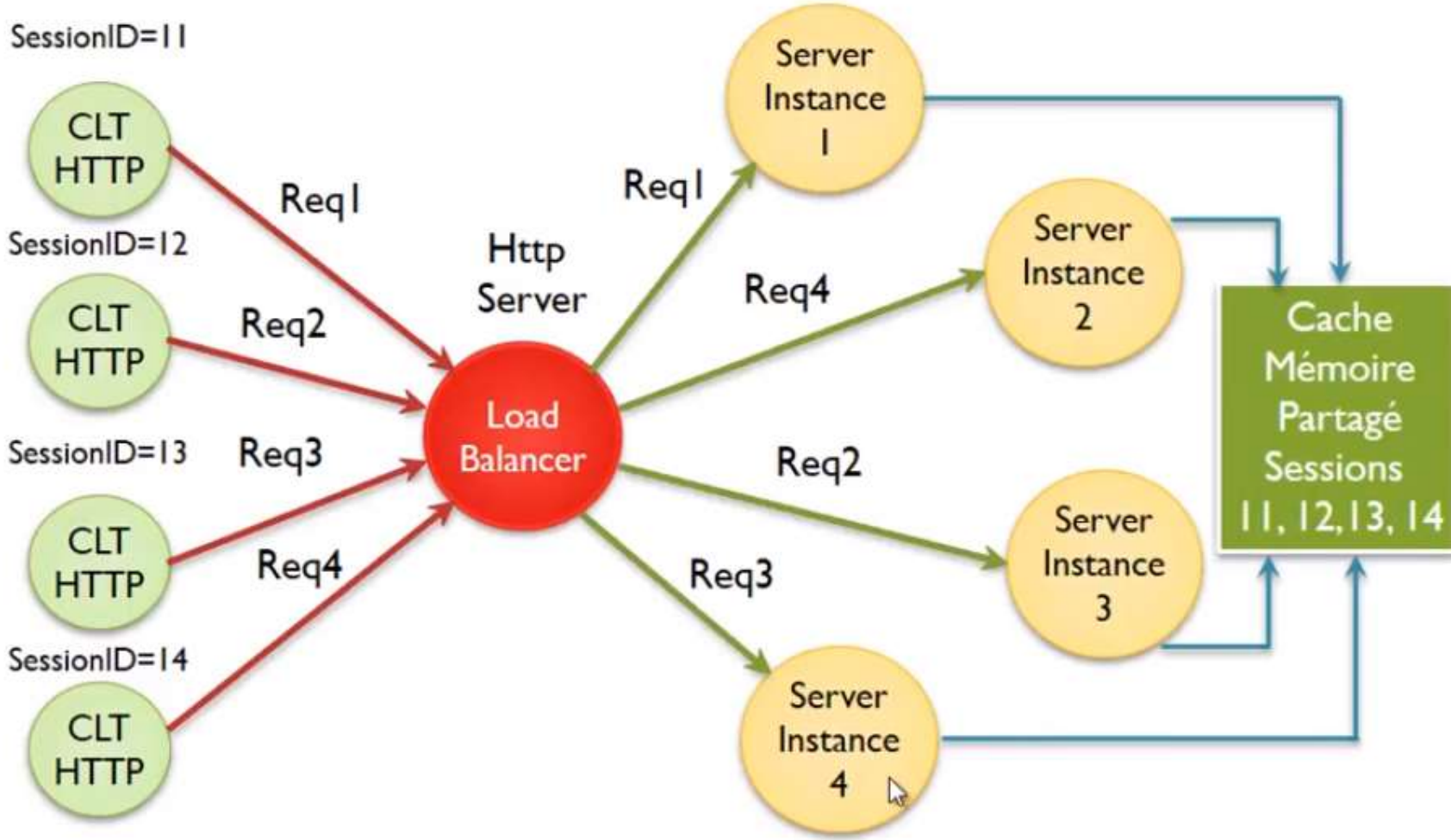
Authentification basée sur les Sessions :

Problème de montée en charge

Cache Distribué et Sticky Sessions



Cache Partagé pour les sessions

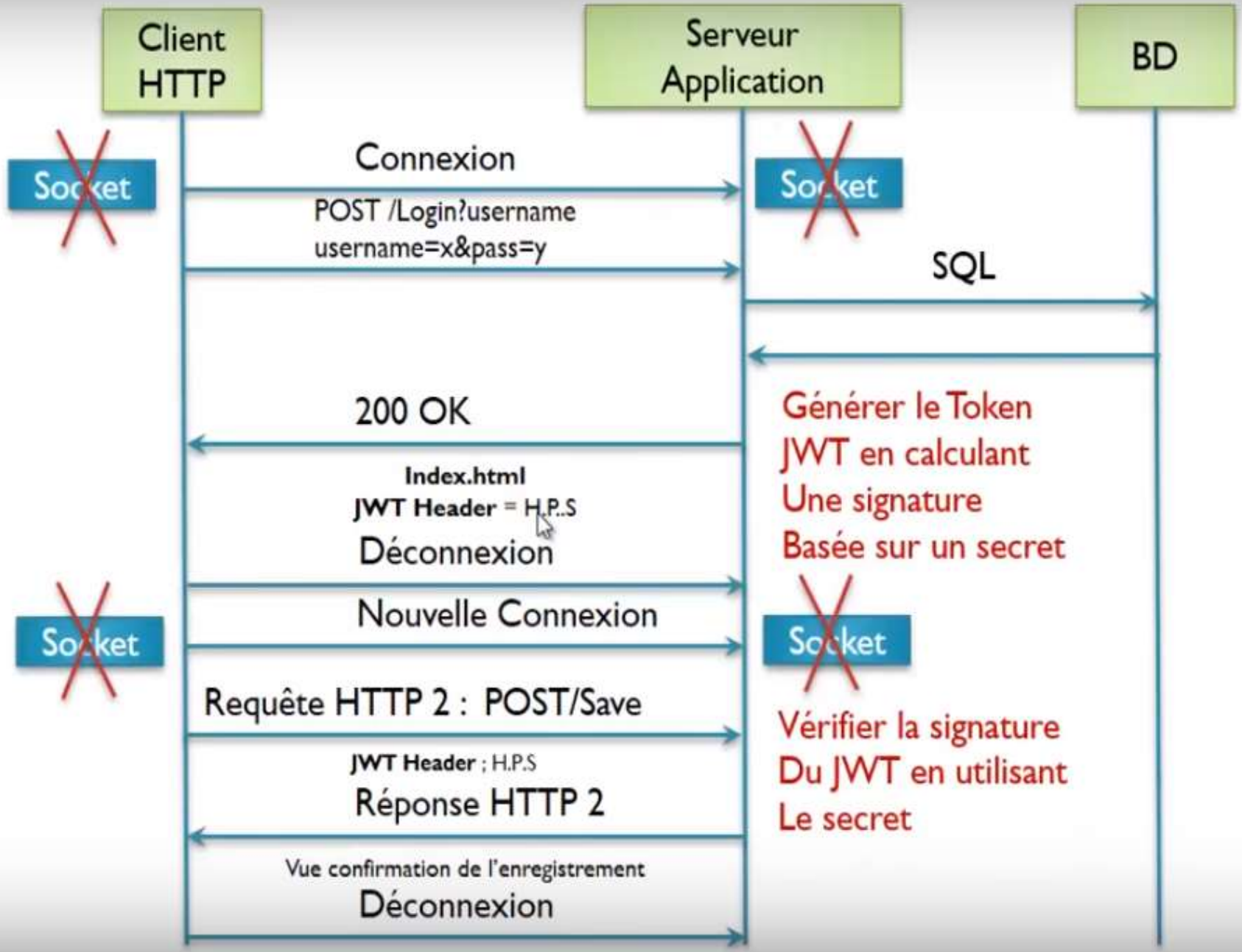


Est-ce qu'il n'y a pas plus simple

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

JWT

z Sécurité Web et Mobile - JSON Web Token



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 connaitre 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 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

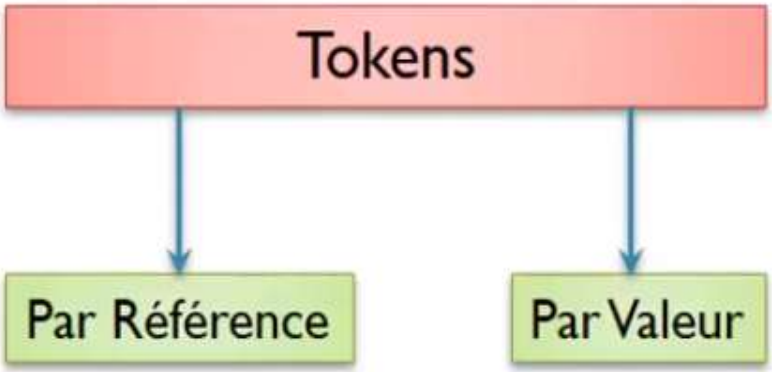
JWT : Signature

- 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

- 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.

Session ID Vs JWT



Session ID = 13

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

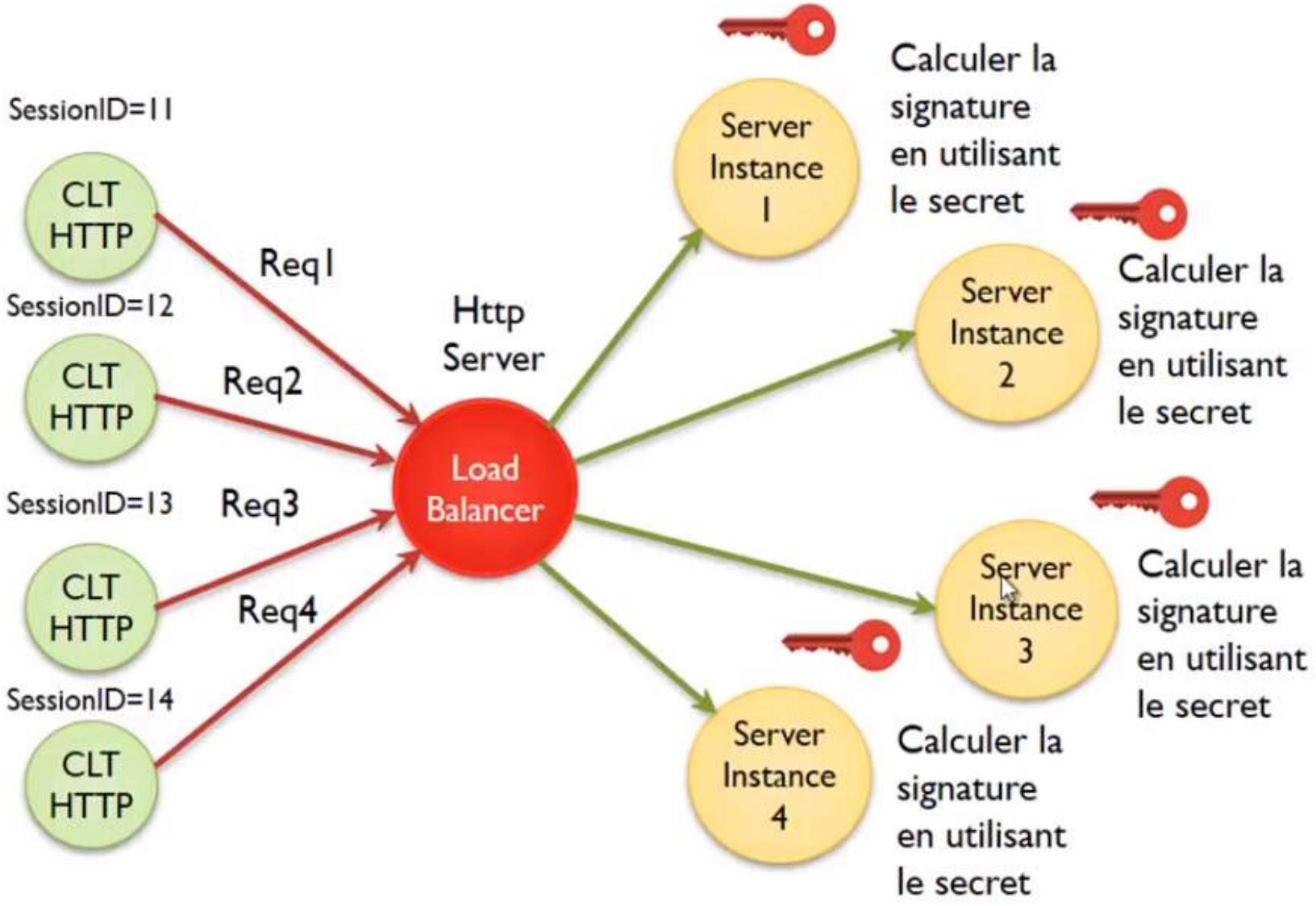
JWT

- 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

Implémentations multi langages

- Java
- PHP
- Java Script
- Pyton
- .Net
- Etc ….

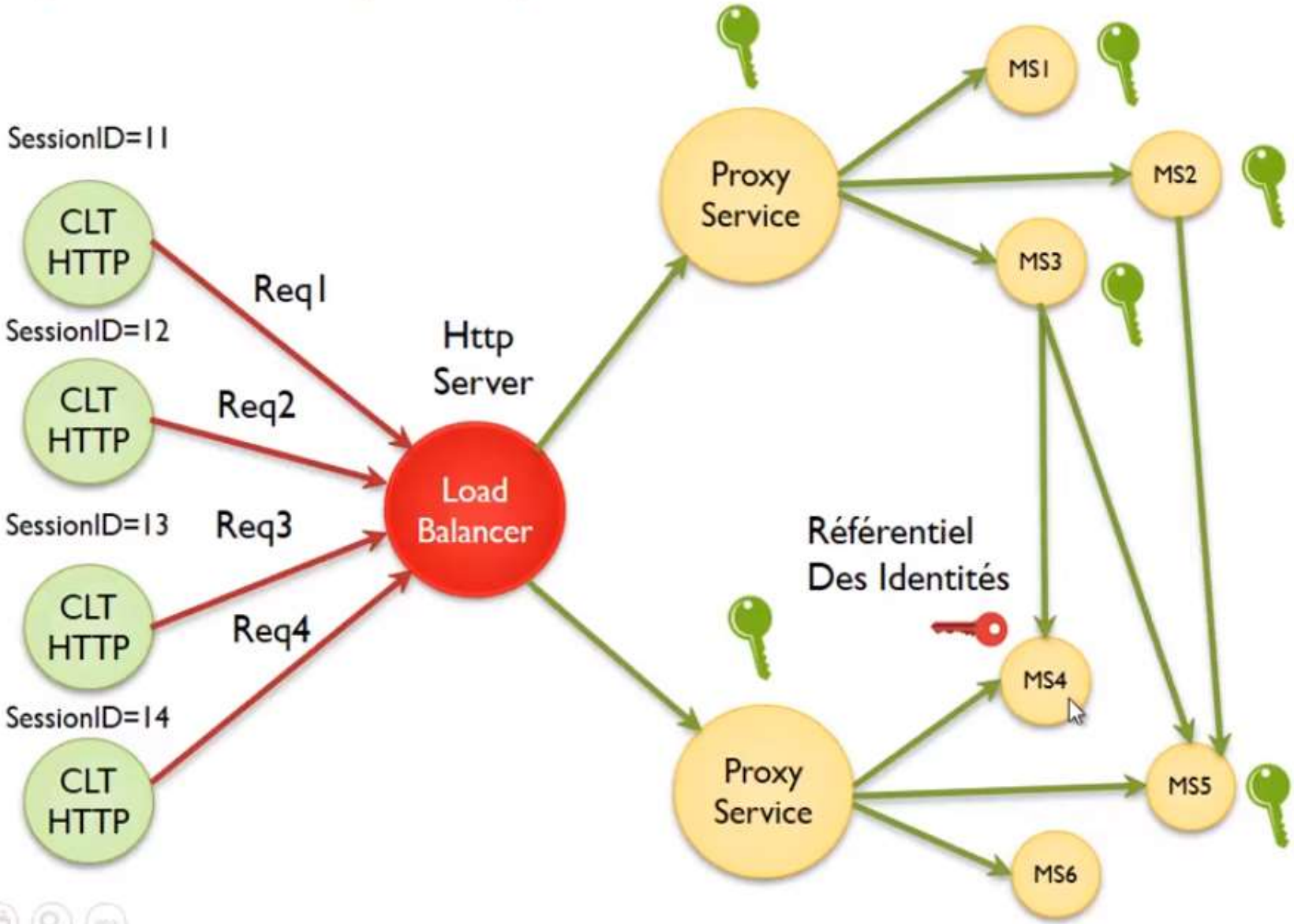
Problème de montée en charge.
Pas de besoin de cache partagé ou distribué



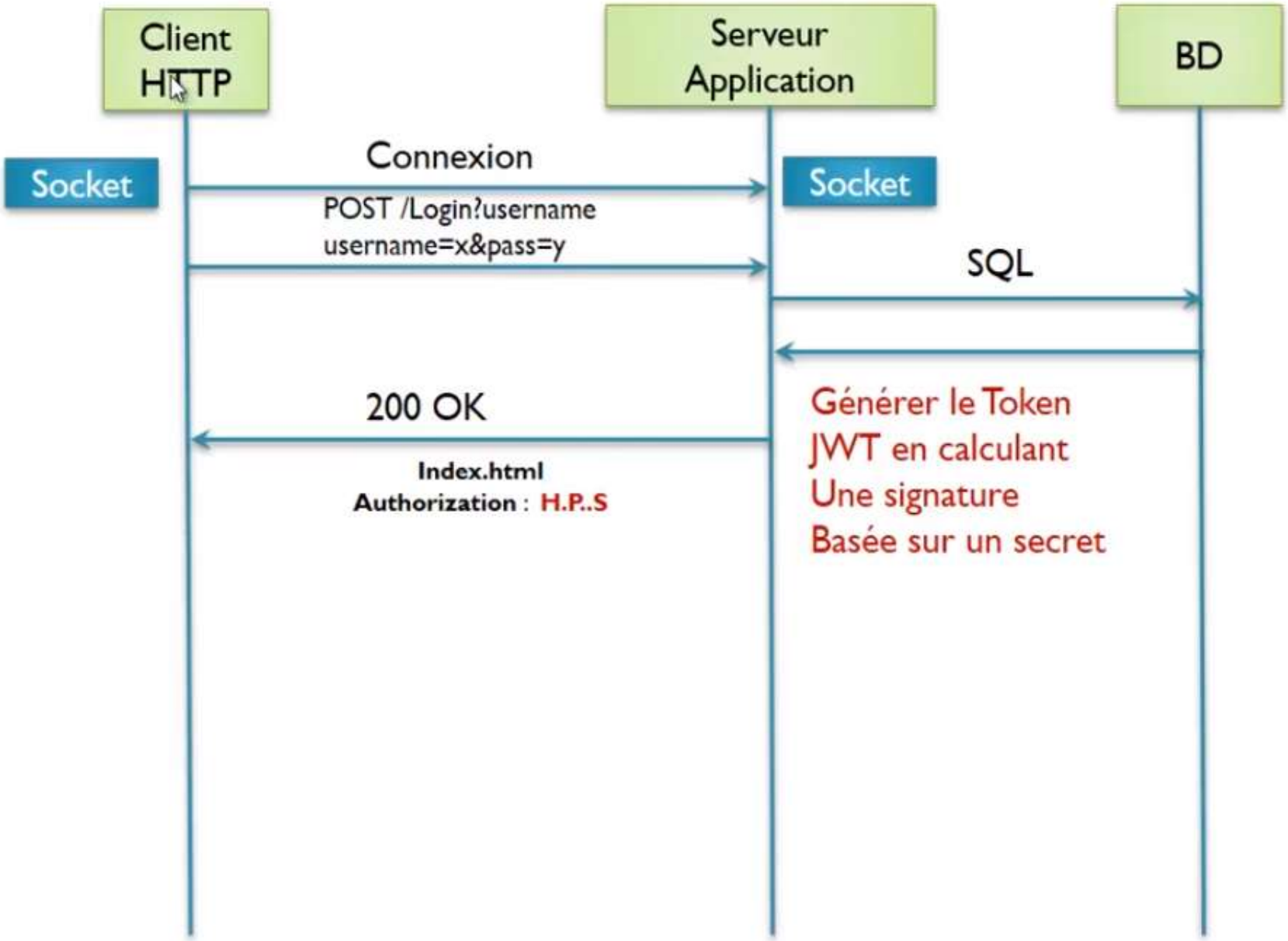
Secret partagé?

- 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

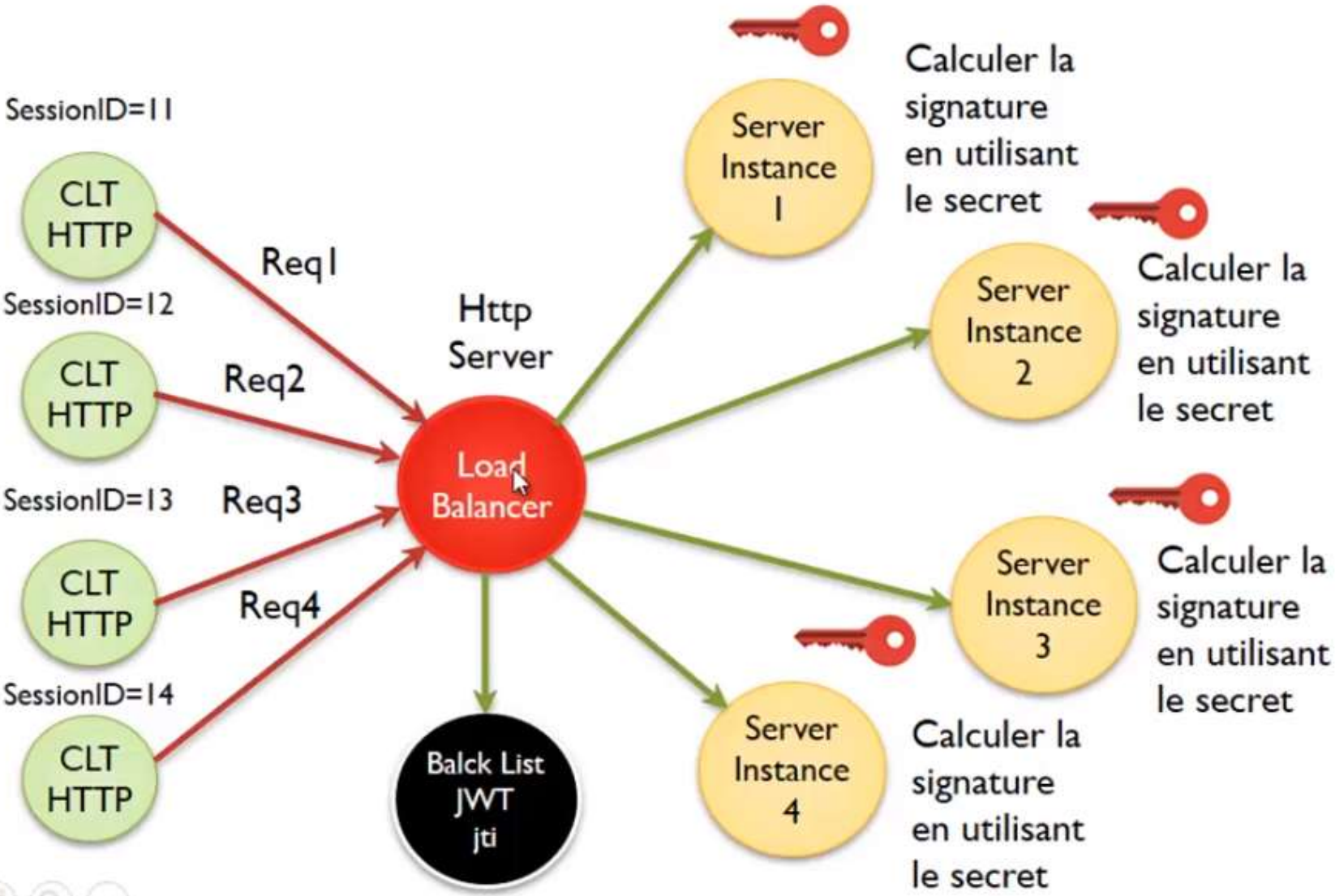
JWT dans un système distribué avec clé privé et clé publique



JWT


Client HTTP
Serveur Application
BD
Socket
Connexion
Socket
POST /Login?username
username=x&pass=y
SQL
200 OK
Index.html
Authorization : H.P.S
Générer le Token
JWT en calculant
Une signature
Basée sur un secret

Révocation des Tokens



Ou 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.

Ou 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 solution 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;HTTPOonly
```

Sécuriser les JWT

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

Autres usages des JWT

- Formulaires en plusieurs parties
  - On peut stocker, coté 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 coté 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 coté serveur pour accomplir cette opération.

Application :

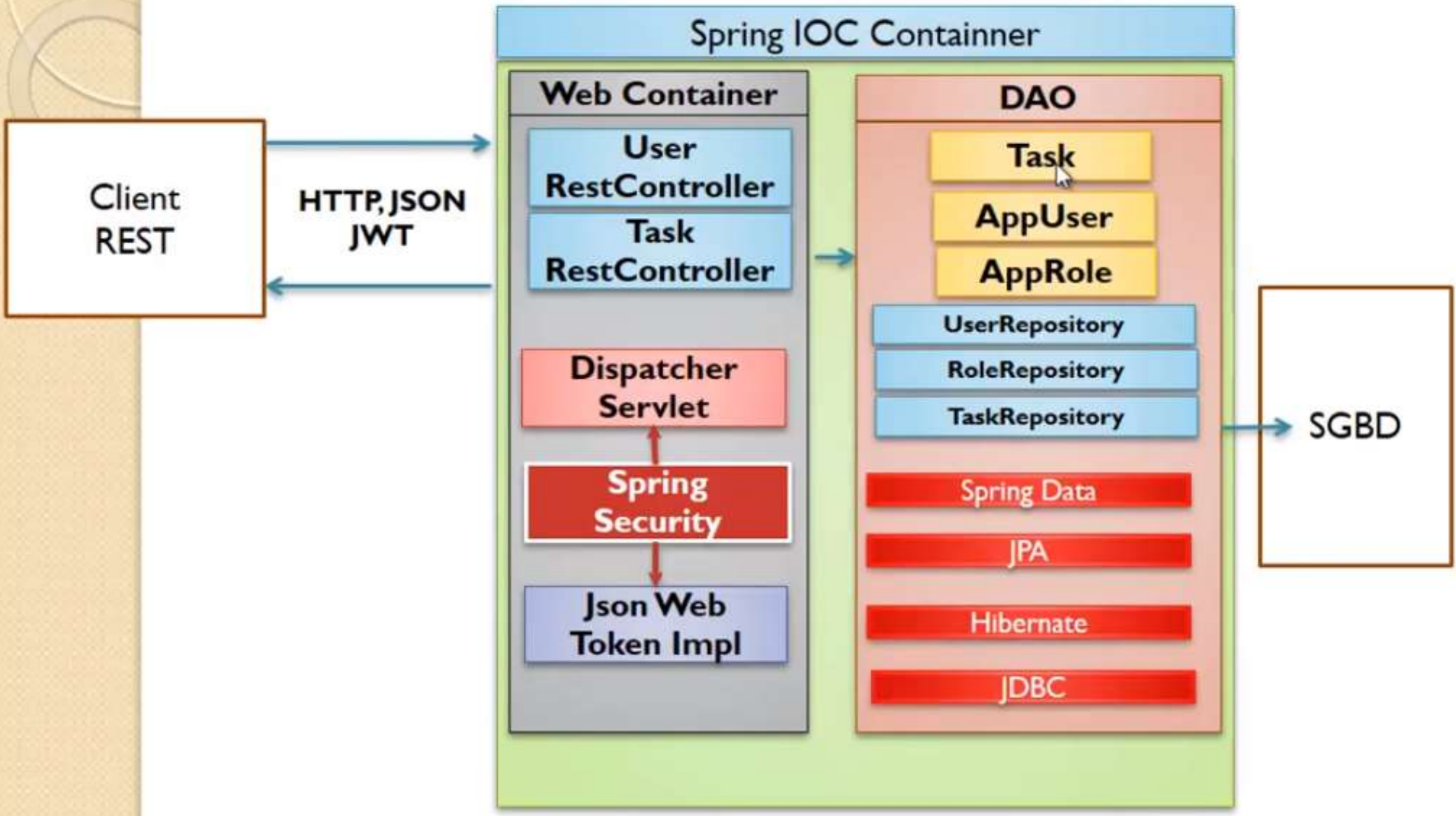
JWT et Spring Security

med@youssfi.net | ENSET Université Hassan II

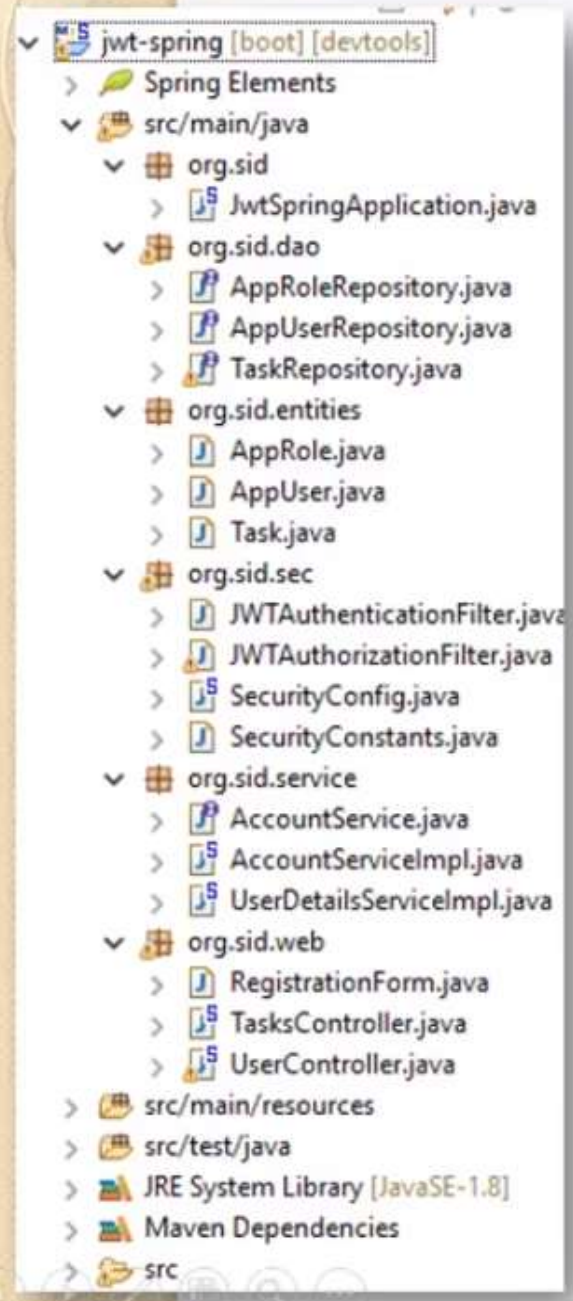
Spring Security et JWT

- 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

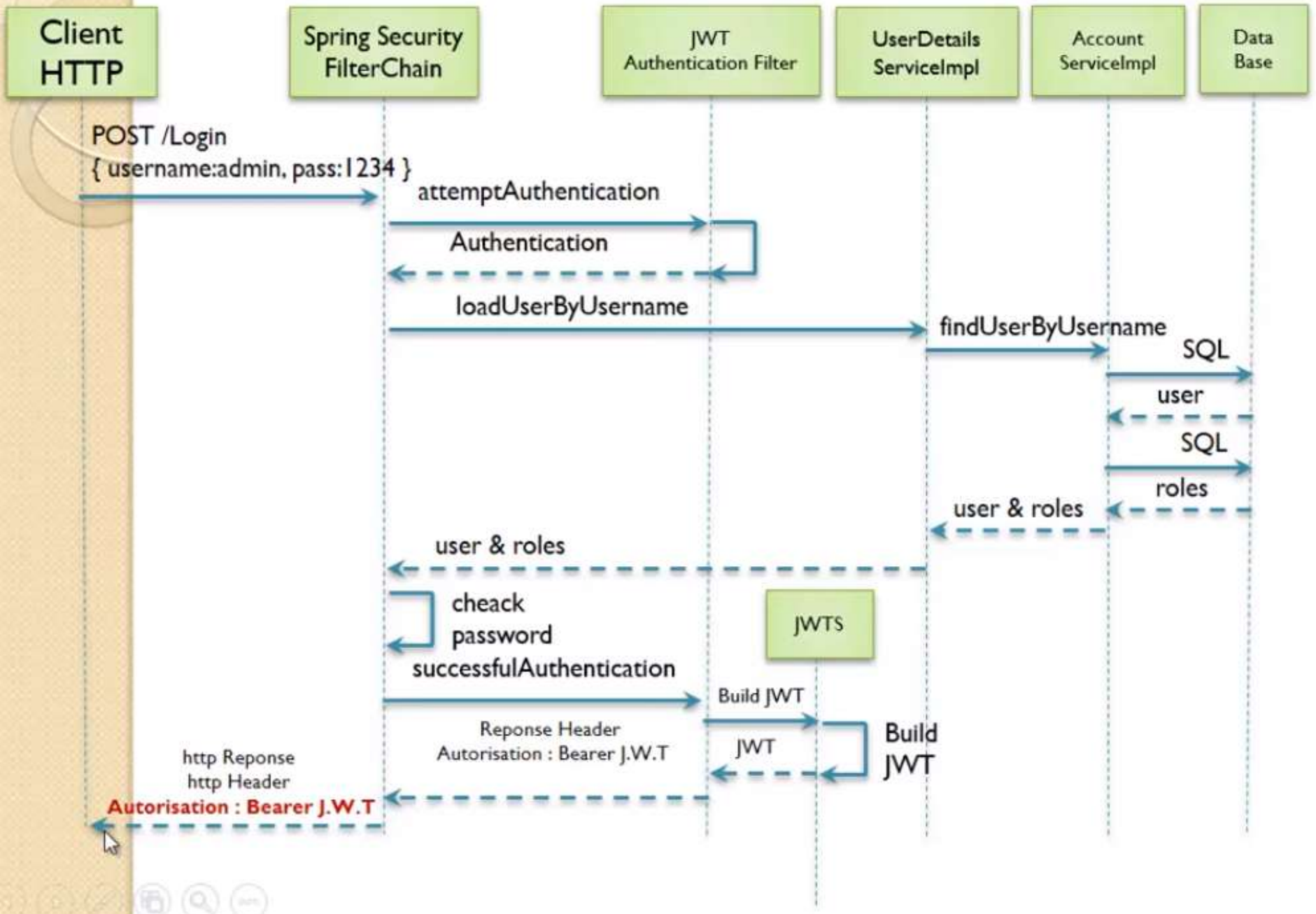


Structure du Projet




```
<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>
```

Authentification Spring Security avec JWT



Demander une ressource nécessitant l'authentification