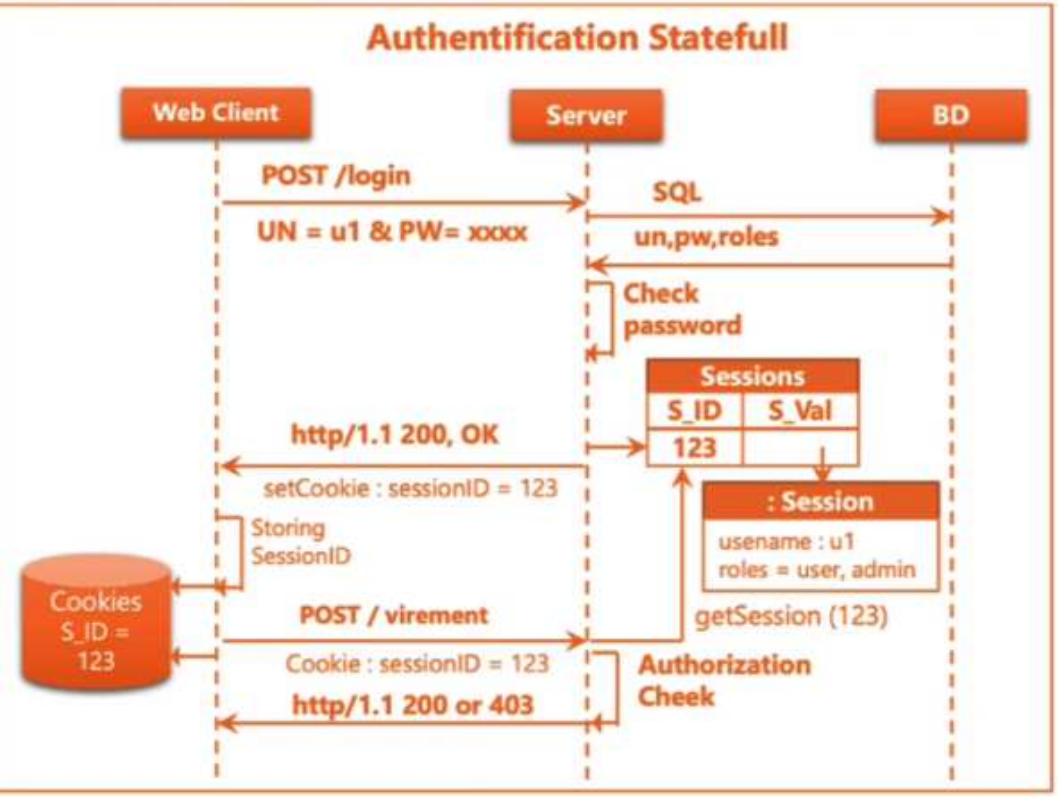
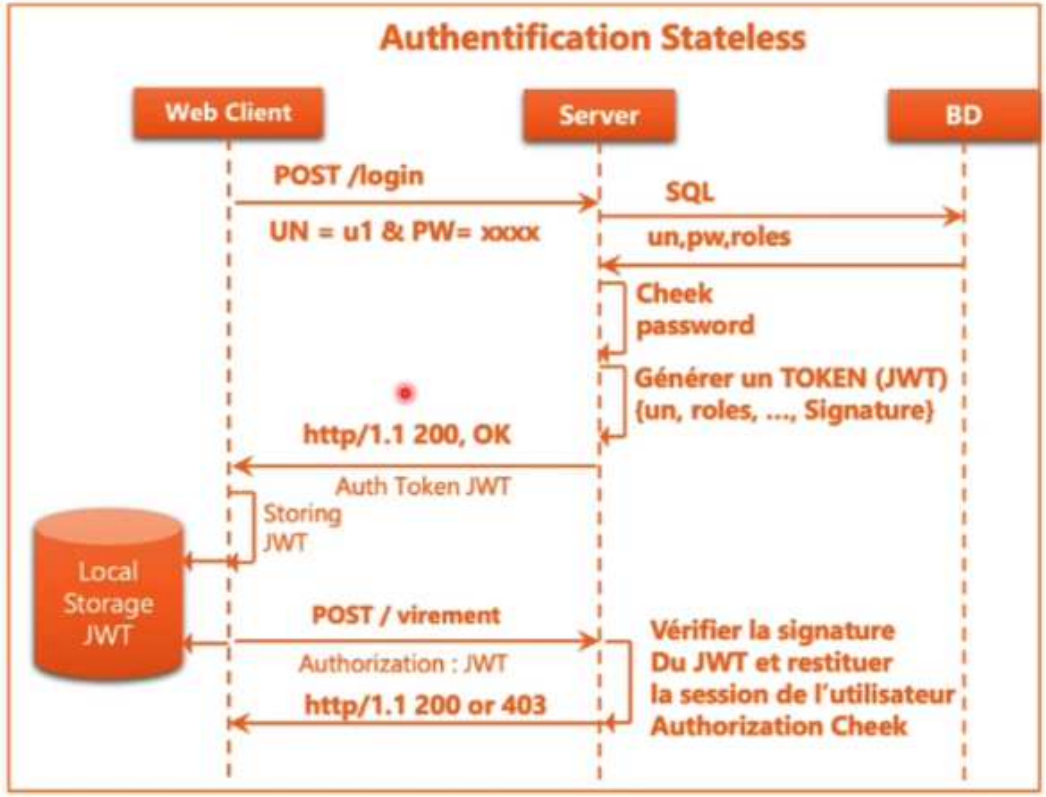
Spring Security

Systèmes d'authentification

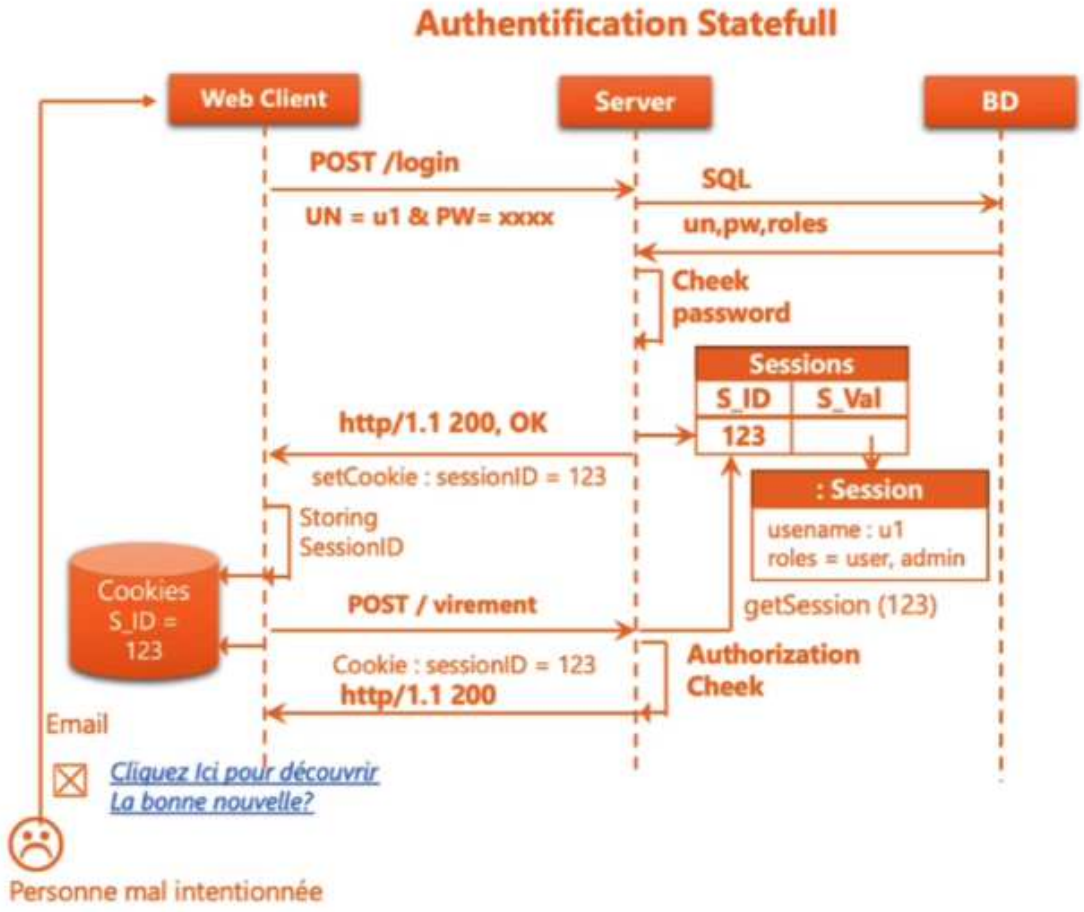
- Deux types de modèles d'authentification :
- **Statefull** : Les données de la session sont enregistrées coté serveur d'authentification
- **Stateless** : les données de la session sont enregistrées dans un jeton d'authentification délivré au client.





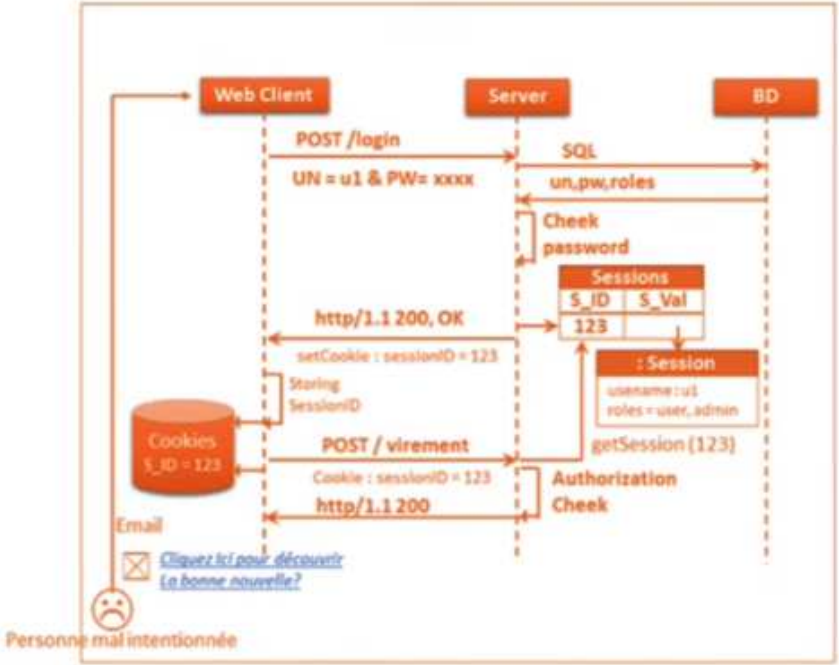
Quelques failles de sécurité : Cross Site Request Forgery (CSRF)

- 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.



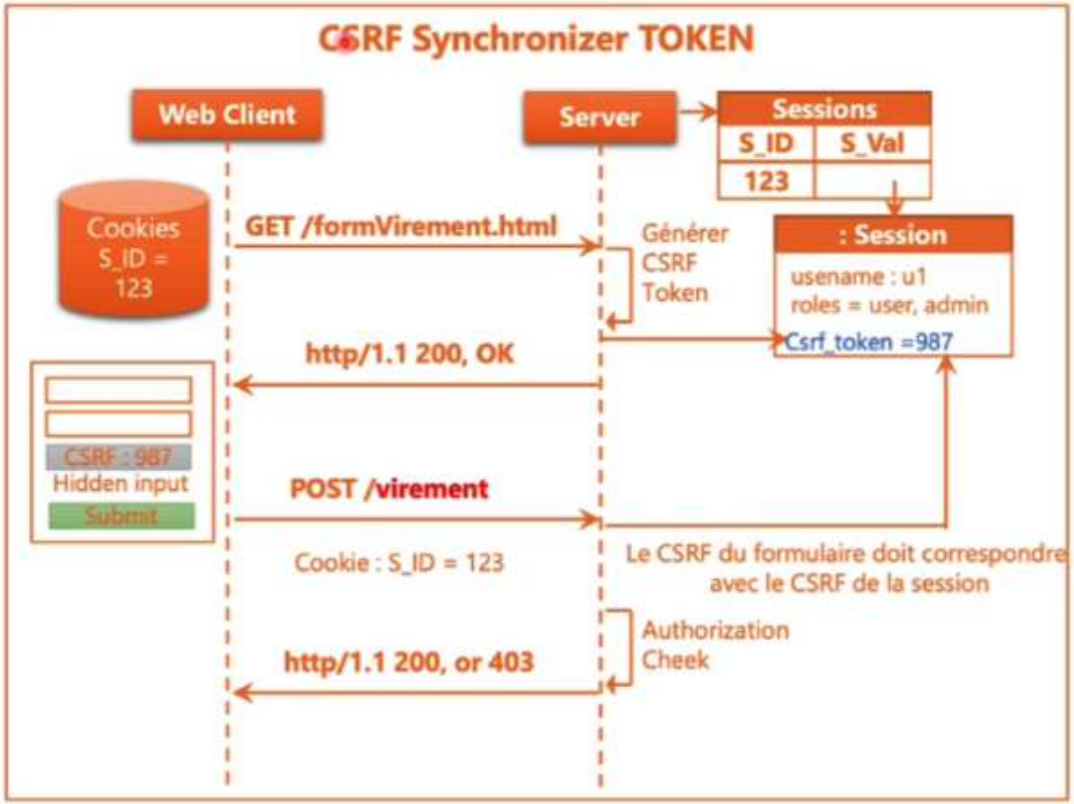
Quelques failles de sécurité : Cross Site Request Forgery (CSRF)

- 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, 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.

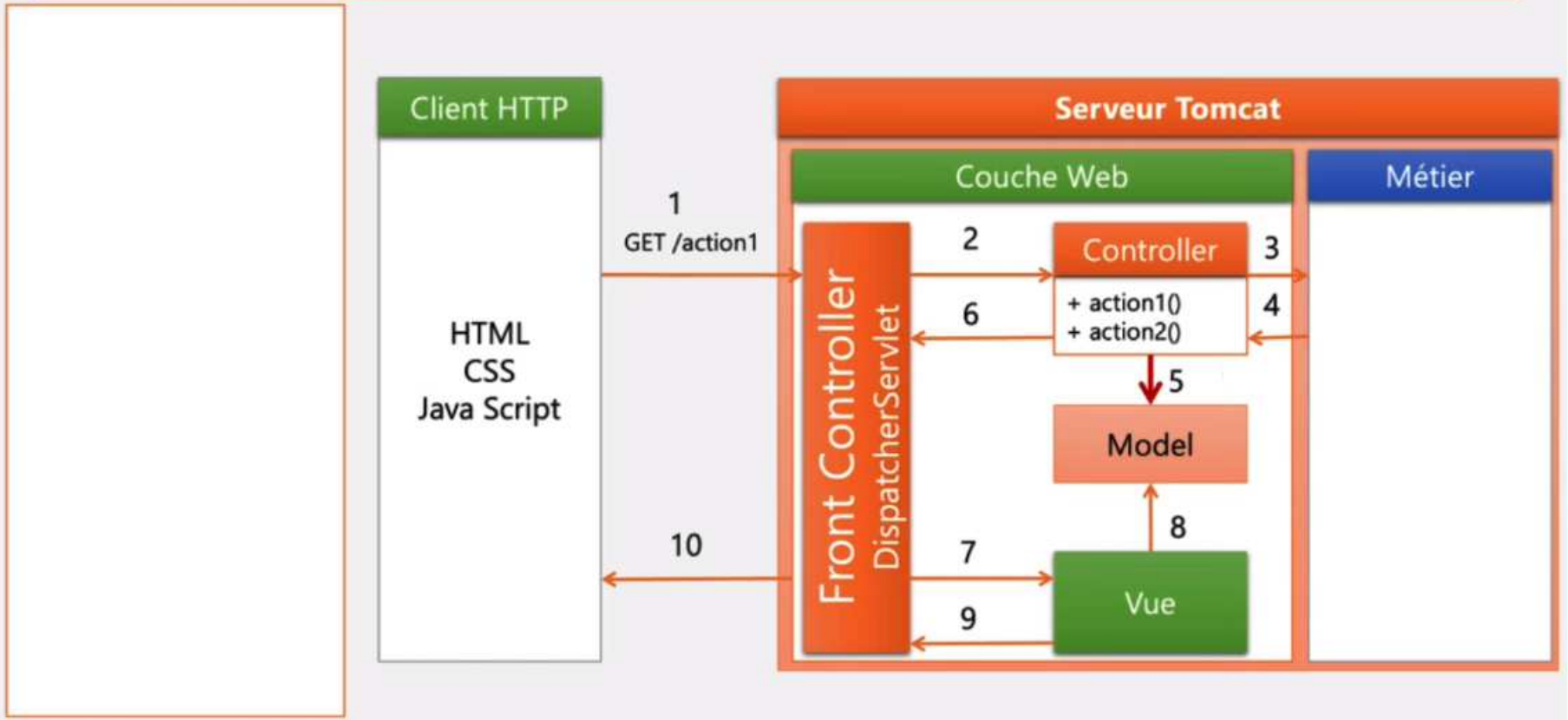


Préventions contre les attaques CSRF

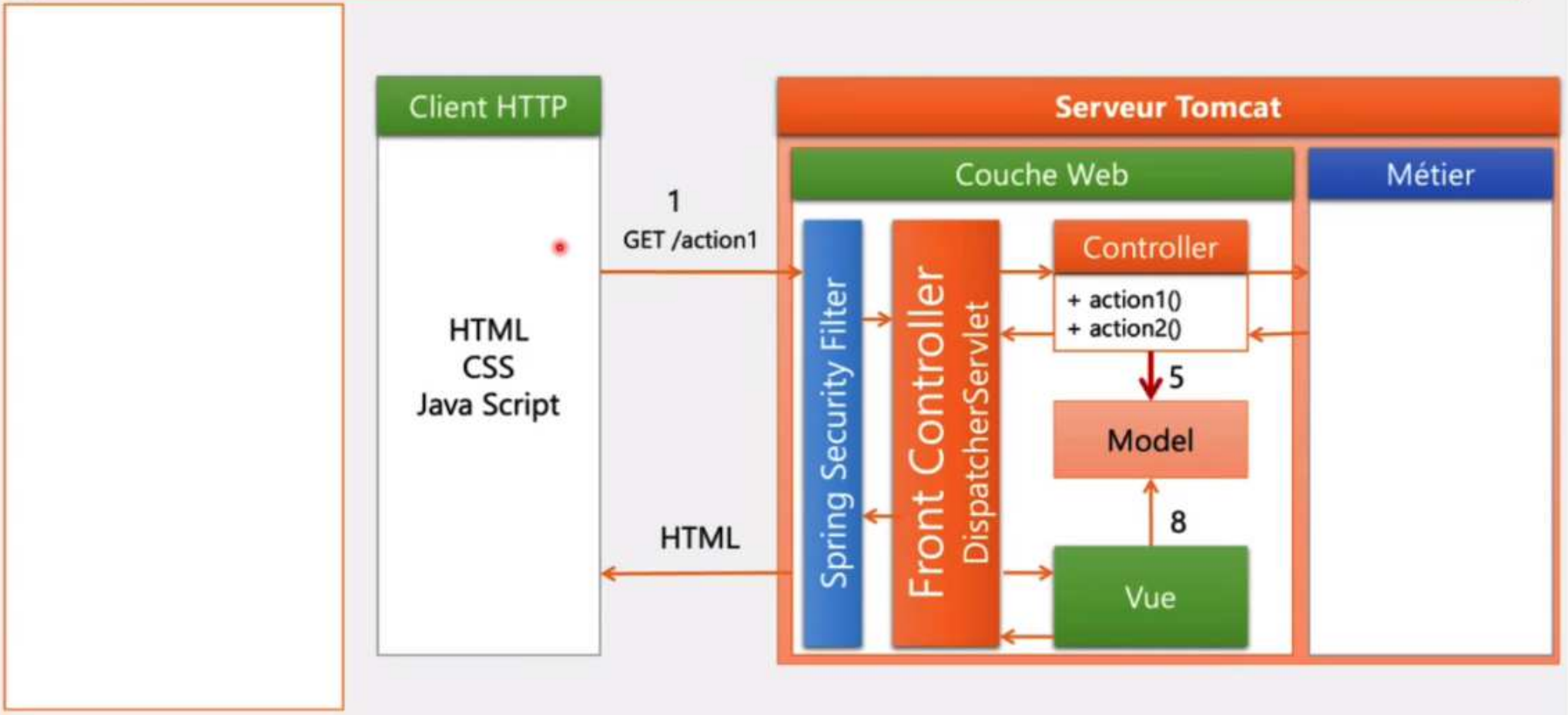
- Utiliser des jetons de validité (CSRF Synchronizer Token) dans les formulaires :
- Faire en sorte qu'un formulaire posté ne soit accepté que s'il a été produit quelques minutes auparavant. Le jeton de validité en sera la preuve.
- Le jeton de validité doit être transmis souvent en paramètre (Dans un 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.



Architecture Spring MVC



Architecture Spring MVC



Configuration de Spring Security

- Spring Security est un module de Spring qui permet de sécuriser les applications Web.
- Spring Security configure des filtres (springSecurityFilterChain) qui permet d'intercepter les requêtes HTTP et de vérifier si l'utilisateur authentifié dispose des droits d'accès à la ressource demandée.
- Les actions du contrôleur ne seront invoquées que si l'utilisateur authentifié dispose de l'un des rôles attribués à l'action.
- Spring Security peut configurer les rôles associés aux utilisateurs en utilisant différentes solution (Mémoire, Base de Données SQL, LDAP, etc..)

Configuration de Spring Security

- Dans cet exemple , nous supposons que nous avons trois tables dans la base de données : users, roles et users_roles et que les mot de passes sont cryptés en format MD5.

roles

role
USER
ADMIN

users

username	password	active
user	ee11cbb19052e40b07aac0ca060c23ee	1
admin	21232f297a57a5a743894a0e4a801fc3	1

users_roles

username	role
admin	ADMIN
admin	USER
user	USER

Dépendance Maven

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

```xml
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
    <version>3.1.0.M1</version>
</dependency>
```

Authentification avec la stratégie « In Memory Authentication »

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Autowired
    PasswordEncoder pwdEncod;

    @Bean
    InMemoryUserDetailsManager inMemoryUserDetailsManager() {
        String pwd = pwdEncod.encode("1234");
        InMemoryUserDetailsManager inMemoryUserDetailsManager =new InMemoryUserDetailsManager(
                User.withUsername("user1").password(pwd).roles("user").build(),
                User.withUsername("admin1").password(pwd).roles("admin").build()
                );
        System.out.println(pwd);
        return inMemoryUserDetailsManager;
    }

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        return httpSecurity.build();
    }

}
```

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Autowired
    PasswordEncoder pwdEncod;

    @Bean
    InMemoryUserDetailsManager inMemoryUserDetailsManager() {
        String pwd = pwdEncod.encode("1234");
        InMemoryUserDetailsManager inMemoryUserDetailsManager =new InMemoryUserDetailsManager(
                User.withUsername("user1").password(pwd).roles("user").build(),
                User.withUsername("admin1").password(pwd).roles("user","admin").build()
                );
        System.out.println(pwd);
        return inMemoryUserDetailsManager;
    }

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
        httpSecurity.authorizeHttpRequests().requestMatchers("/delete").hasRole("admin");
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        return httpSecurity.build();
    }

}
```

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
public class SecurityConfig {
    @Autowired
    PasswordEncoder pwdEncod;

    @Bean
    InMemoryUserDetailsManager inMemoryUserDetailsManager() {
        String pwd = pwdEncod.encode("1234");
        InMemoryUserDetailsManager inMemoryUserDetailsManager =new InMemoryUserDetailsManager(
                User.withUsername("user1").password(pwd).roles("user").build(),
                User.withUsername("admin1").password(pwd).roles("user","admin").build()
                );
        System.out.println(pwd);
        return inMemoryUserDetailsManager;
    }

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
        //httpSecurity.authorizeHttpRequests().requestMatchers("/delete").hasRole("admin");
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        return httpSecurity.build();
    }
```

```java
@PreAuthorize("hasRole('admin')")
@GetMapping("/delete")
public String removeProduct(Model model, @RequestParam("ID") Integer id, @RequestParam("criteria")
    pr.deleteById(id);
    return "redirect:/list?criteria=" + mc;
}
```