Sécurisation d'application WEB Spring Security

Mise en oeuvre

Rappel des concepts de sécurité

Contrôle d'accès

- Définition

  *Le contrôle d'accès est l'ensemble des outils de sécurité qui **contrôlent comment** les **utilisateurs** et systèmes **interagissent** avec le **SI** (systèmes et ressources)*

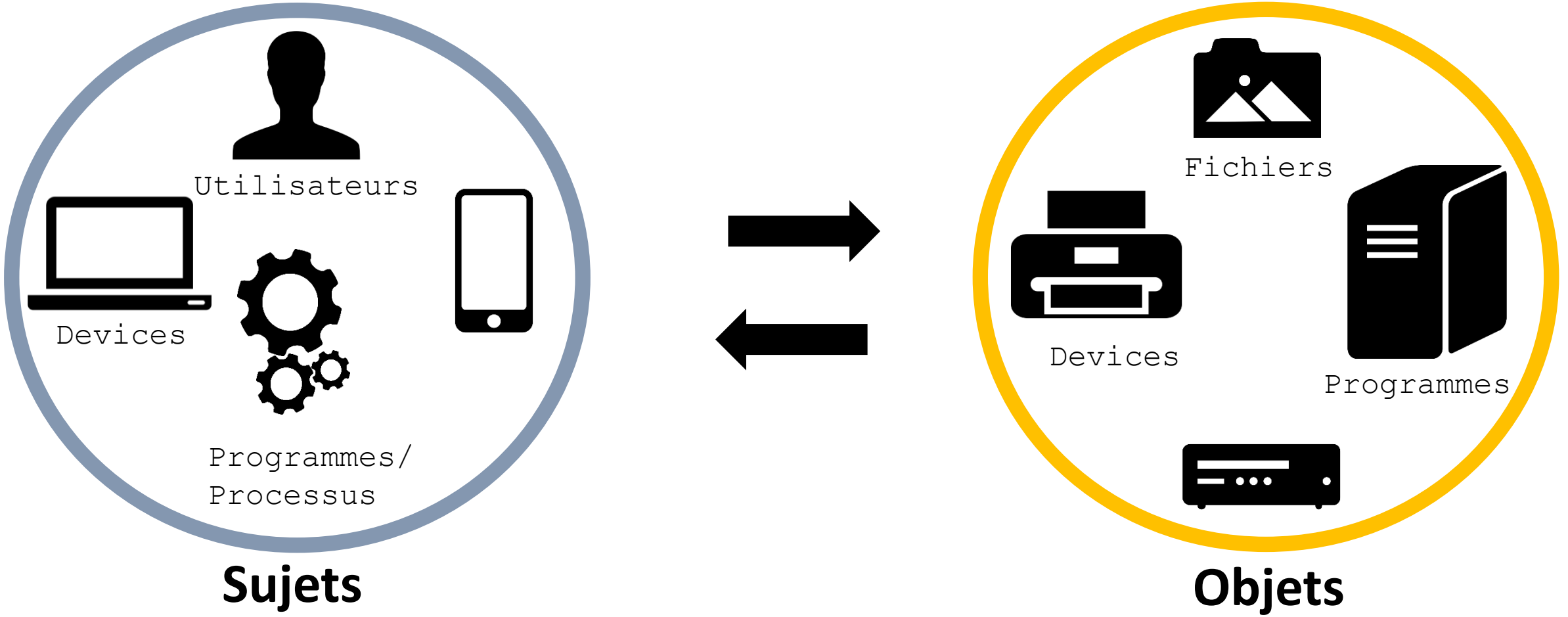
- Les concepts clés
  - **Accès**

    L'accès est le flux d'information entre un sujet et un objet
  - **Sujet**

    Utilisateur, programme, processus qui accède à un objet afin d'accomplir une tâche
  - **Objet**

    Entité passive contenant de l'information.

Contrôle d'accès



Concepts

- Authentication

  Vérifier *l'authenticité de* ***l'identité*** *d'une entité (what you know, what you have, what you are).*

- Authorization

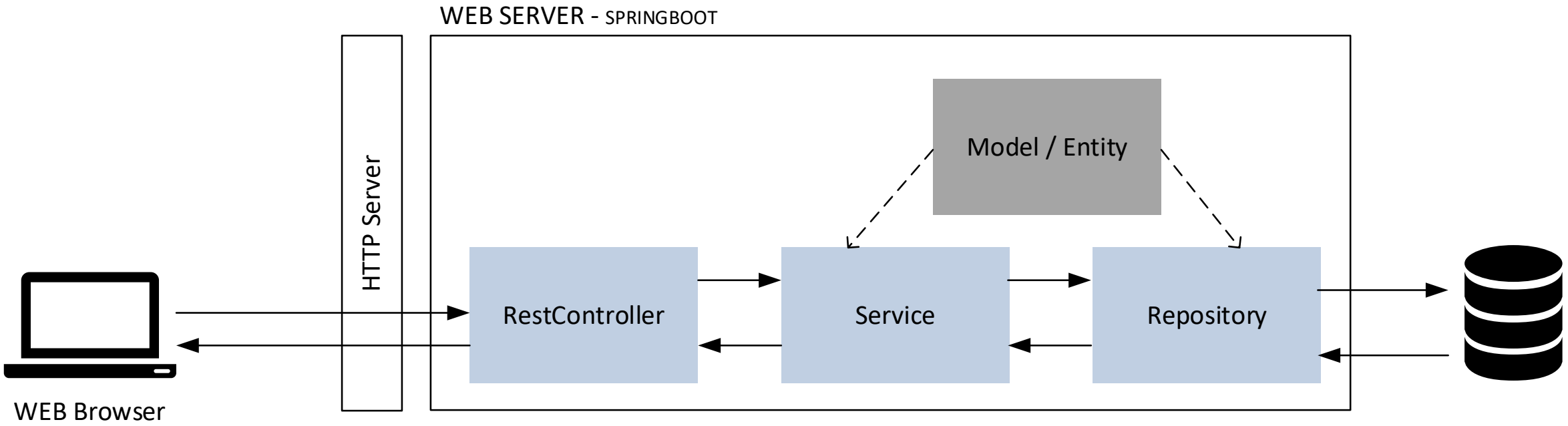
  Assignation *de* ***droits****, autorisation en accord avec la politique de sécurité en vigueur.*

- Filtrage

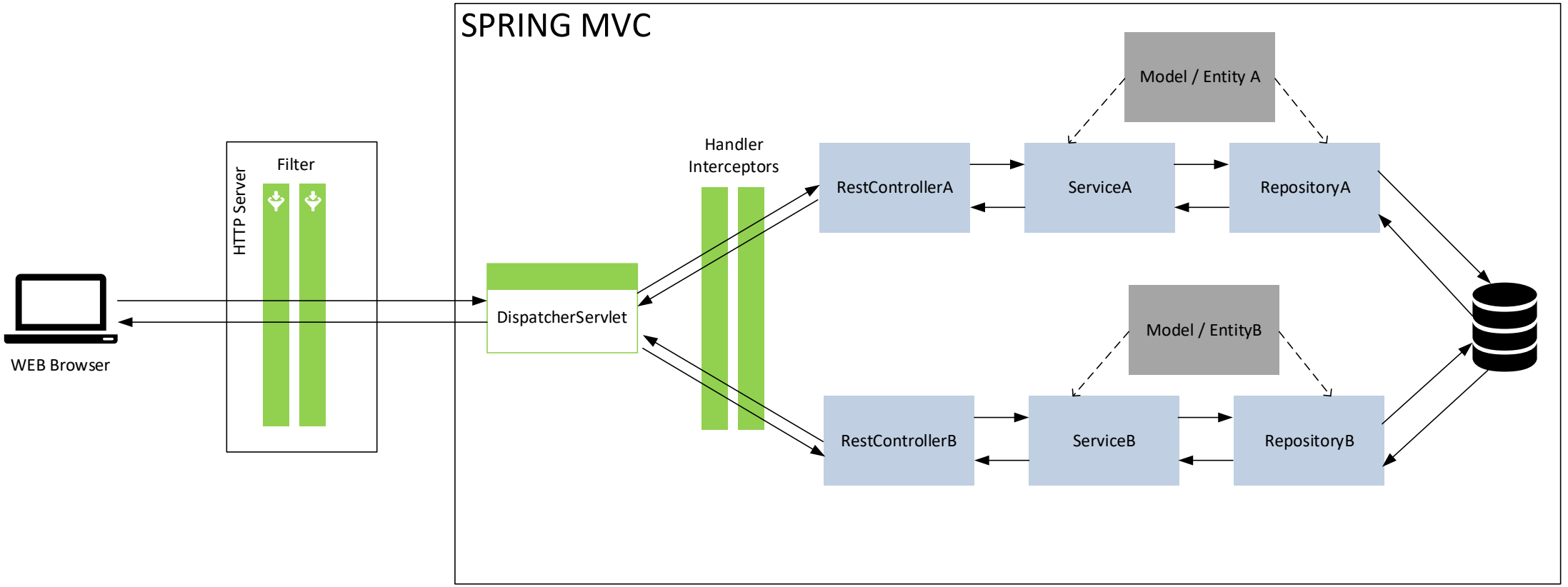
  Mise ne place de ***règles*** *(de différents niveaux OSI) permettant de* ***sélectionner*** *des données pouvant* ***accéder*** *à plusieurs zones de sécurité. (le non respect de règles entraine un blocage)*

Spring Security premiers pas

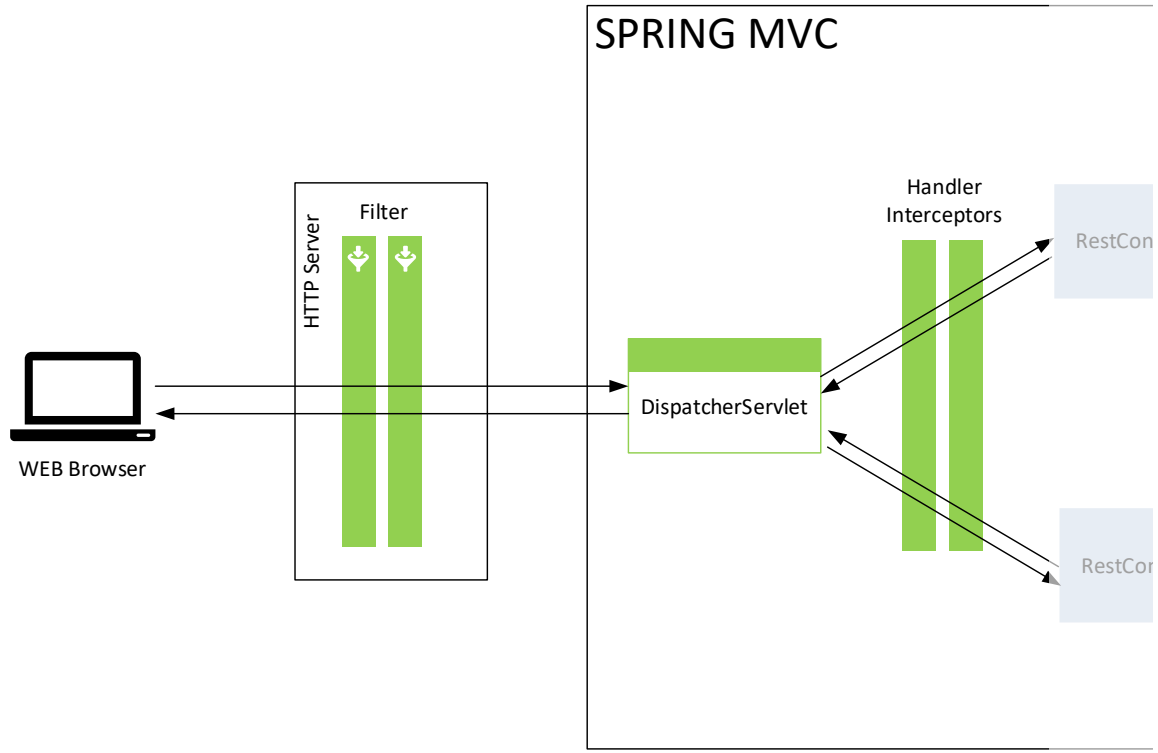
Organisation de Spring

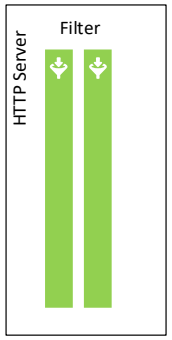


Organisation de Spring



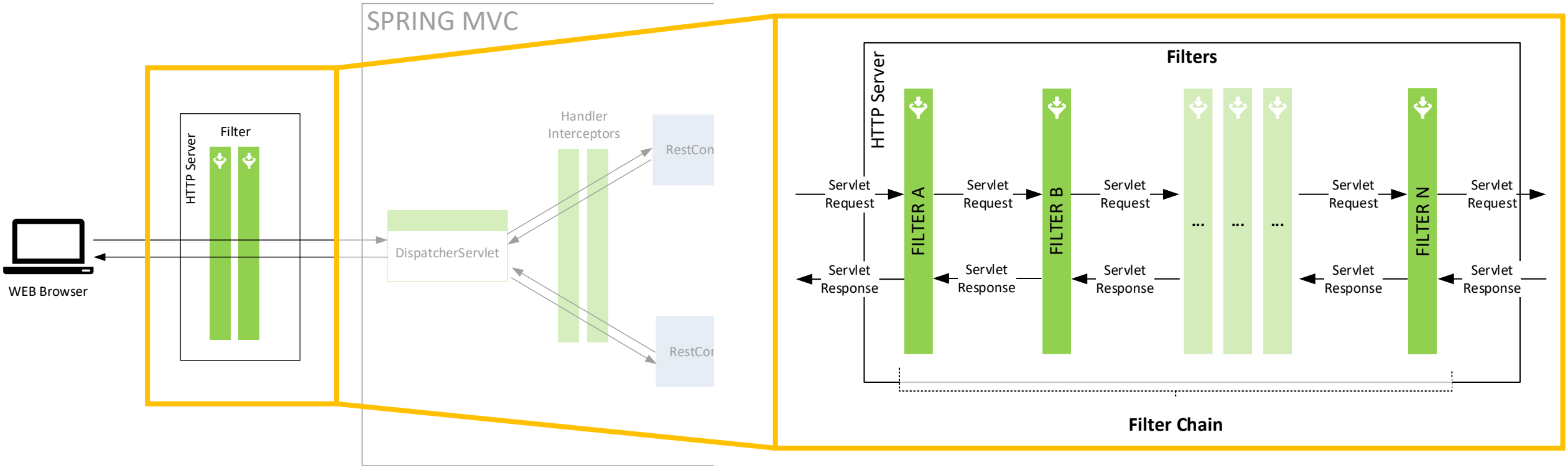
Organisation de Spring



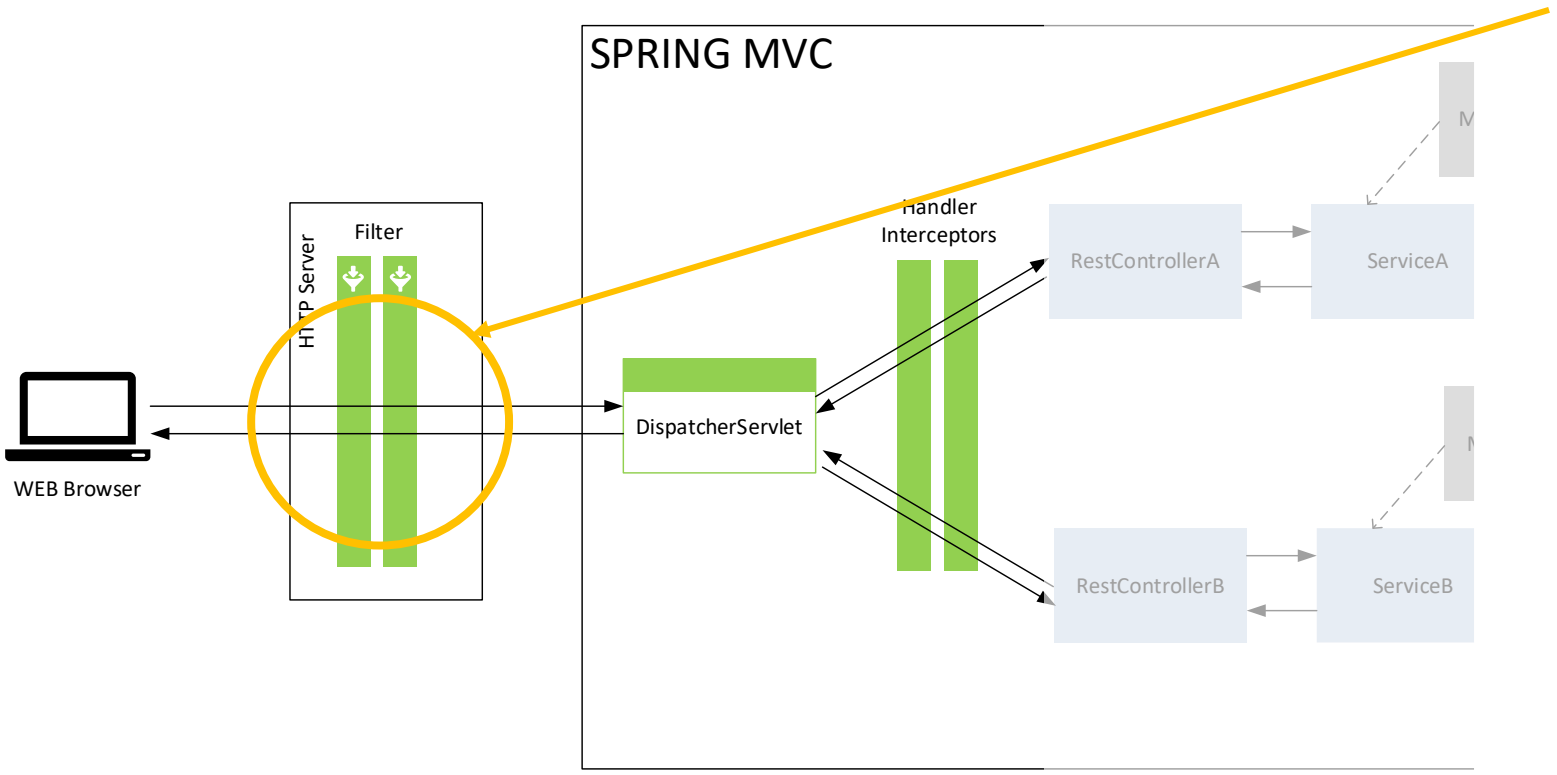


- **Dispatcher** : Servlet qui délègue toutes les requêtes http aux contrôleurs
- **Filter** : Membre du serveur http, utilisés pour manipuler **TOUTES** les requêtes avant le Dispatcher
  - Traitement gros grain des requêtes:
    - Authentication
    - Logging and auditing
    - Image and data compression
    - …
- **HandlerInterceptors** : Membre du framework SpringMVC, intercepte les requêtes entre le dispatcher et les controleurs (idem pour les view)
  - Traitement fins des requêtes:
    - Gestion des tâches transverses (e.g app logging)
    - Verification détaillée des autorisations
    - Manipulation du model ou du context Spring
    - …

Organisation de Spring

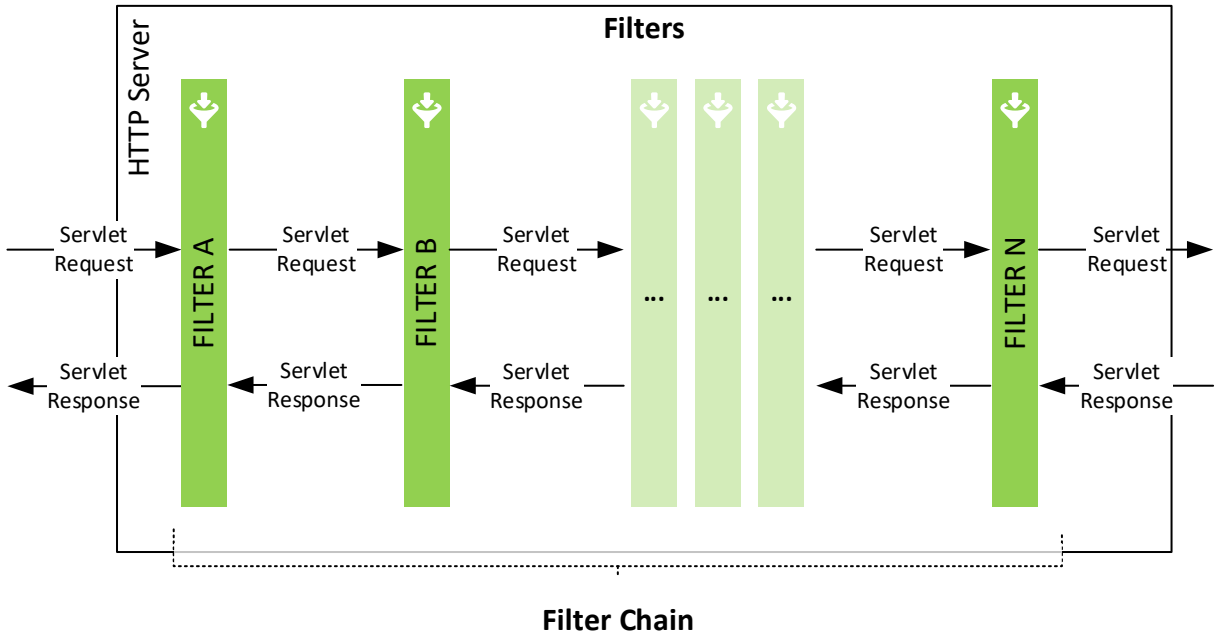


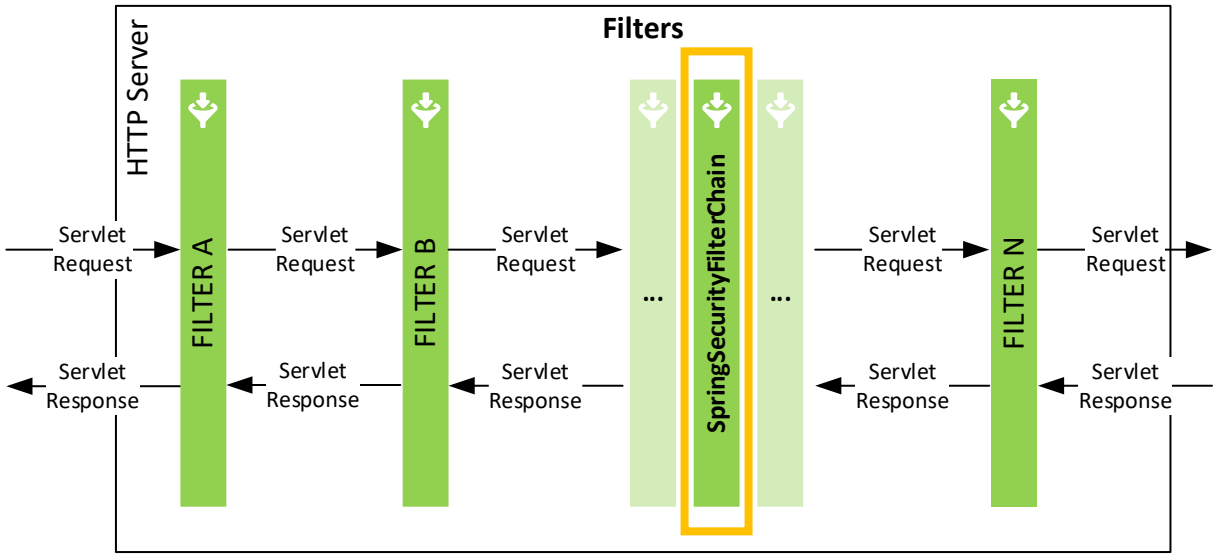
Spring Security



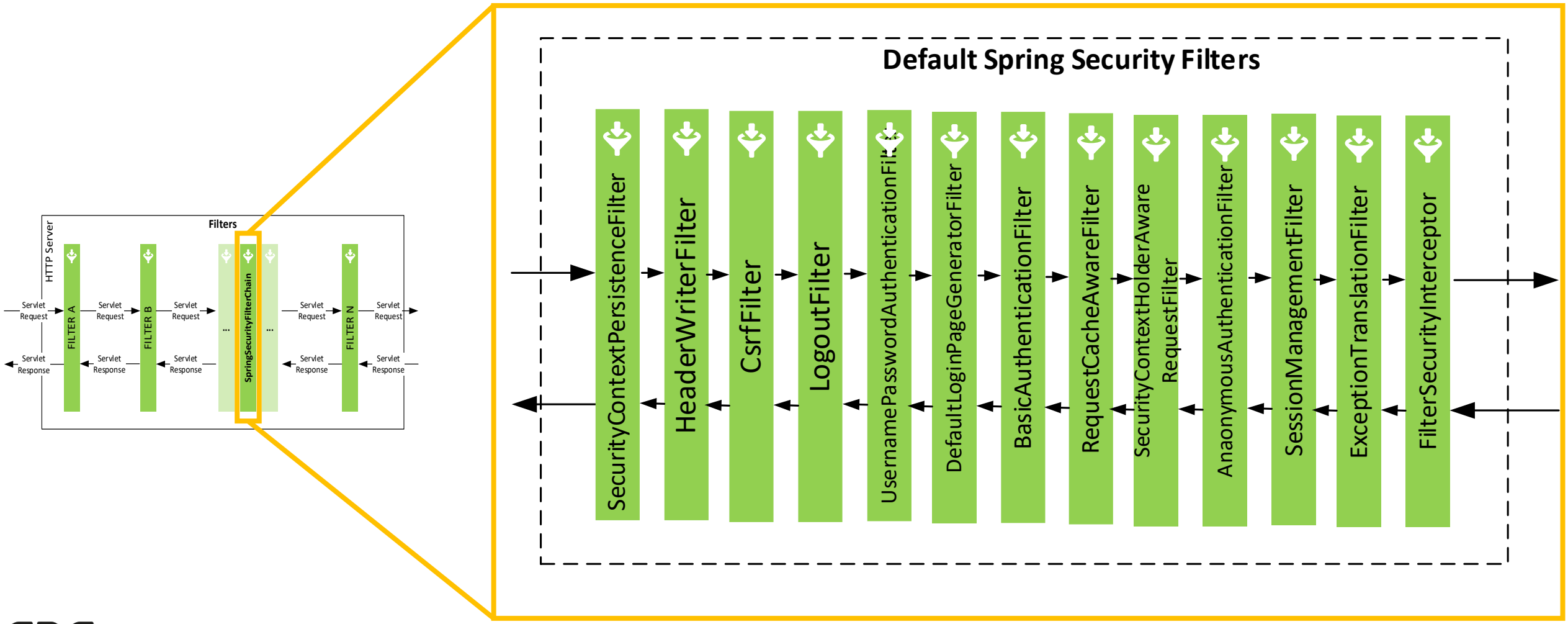
- **Spring Security** : opère principalement au niveau des **Filters** et fournit
  - Authentification
  - Autorisation
  - De la redirection si nécessaire
  - Génération de token
  - …

Spring Security

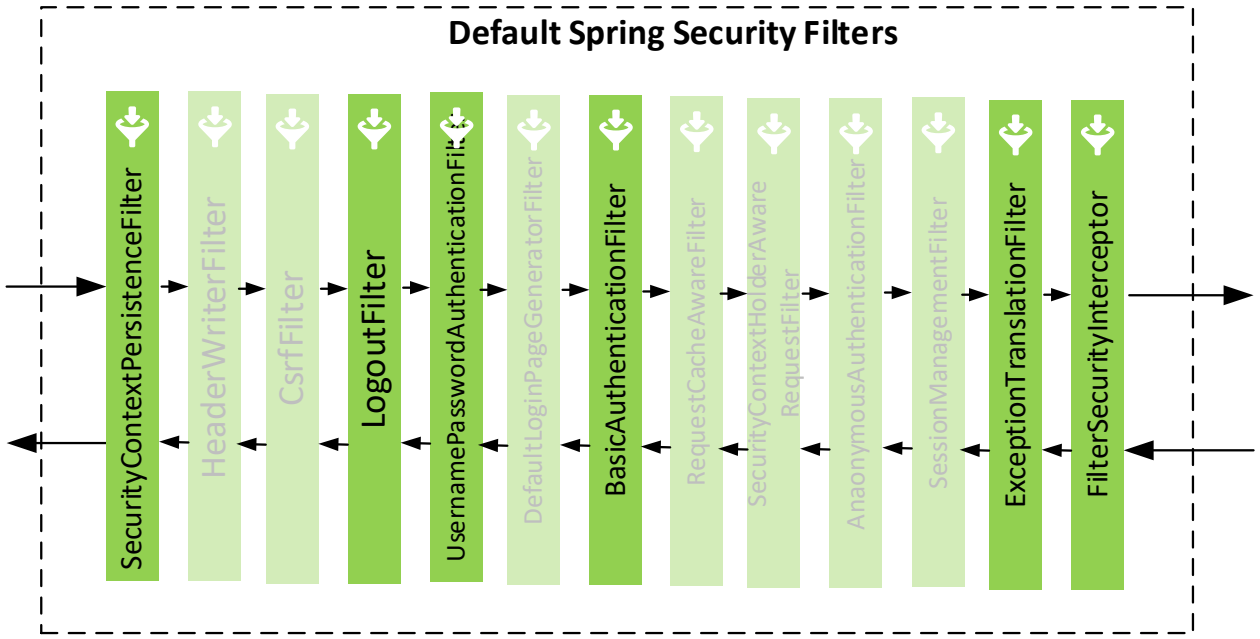




Spring Security



Spring Security



- **SecurityContextPersistenceFilter** : met en place et maintient le SecurityContext entre les requêtes HTTP
- **LogoutFilter** : Nettoie le SecurityContext lors d'un logout d'un utilisateur
- **UsernamePasswordAuthenticationFilter** : tentative d'authentificaiton par login/password dans le champ body (post)
- **BasicAuthenticationFilter**: tentative d'authentification via basic Auth (RFC 2617)
- **ExceptionTranslastionFilter**: Convertit les exceptions SpringSecurity en réponse HTTP ou redirige la requête.
- **FilterSecurityInterceptorFilter**: applique les règles d'autorisation (basées sur la configuration et la notion d'autorité)

Spring Security Authentication

Spring Security

- Première Configuration
  - Mise en place d'une sécurité par default
  - Application d'une chaine de filtrage
  - `@EnableWebSecurity` : active le support de sécurité web
  - `SecurityFilterChain` : Bean permettant de configurer les règles de sécurité sur les Urls spécifiées

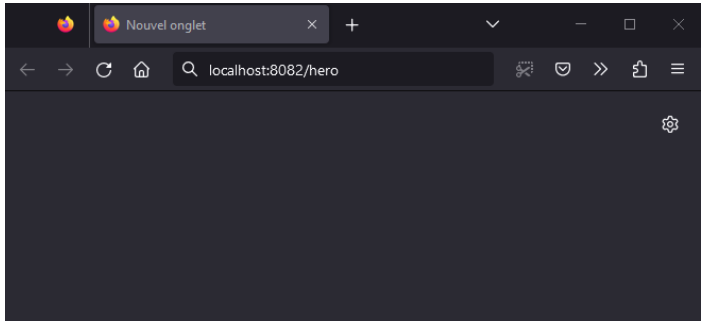
```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
                                                throws Exception {
        http.csrf(csrf->csrf.disable());
        http
                .formLogin(form ->
                        form.loginPage("/login")
                            .permitAll()
                        )
                .authorizeHttpRequests(auth ->
                        auth.requestMatchers("/login")
                            .permitAll())

                .authorizeHttpRequests(auth ->
                        auth.requestMatchers("/hero/**")
                            .authenticated()
                            .anyRequest().permitAll())
                .logout(logout->
                        logout.logoutUrl("/logout")
                              .permitAll()
                              .invalidateHttpSession(true));
        return http.build();
    }
}
```

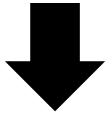
Spring Security

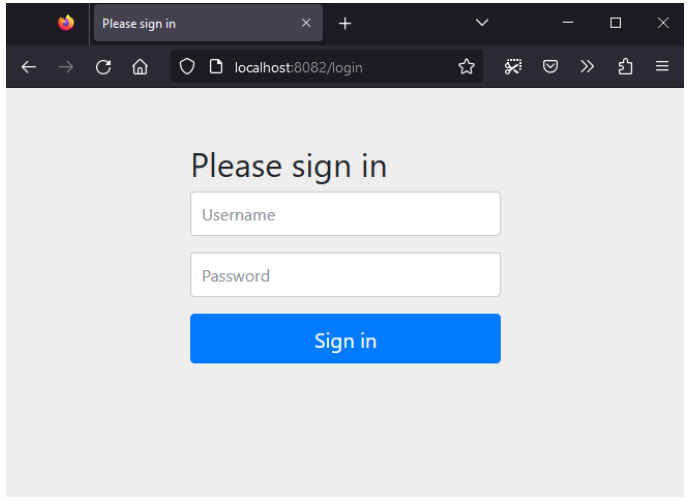
- Première Configuration
  - Mise en place d'une sécurité par default
  - Application d'une chaine de filtrage
  - `requestMatcher` : URL sur laquel porte l'action
  - `permitAll, authenticated` : action
  - `formLogin / FormLogout` formulaire auto généré

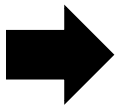
```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
                                                    throws Exception {
        http.csrf(csrf->csrf.disable());
        http
                .formLogin(form ->
                        form.loginPage("/login")
                            .permitAll()
                        )
                .authorizeHttpRequests(auth ->
                        auth.requestMatchers("/login")
                            .permitAll())

                .authorizeHttpRequests(auth ->
                        auth.requestMatchers("/hero/**")
                            .authenticated()
                            .anyRequest().permitAll())
                .logout(logout->
                        logout.logoutUrl("/logout")
                              .permitAll()
                              .invalidateHttpSession(true));
        return http.build();
    }
}
```

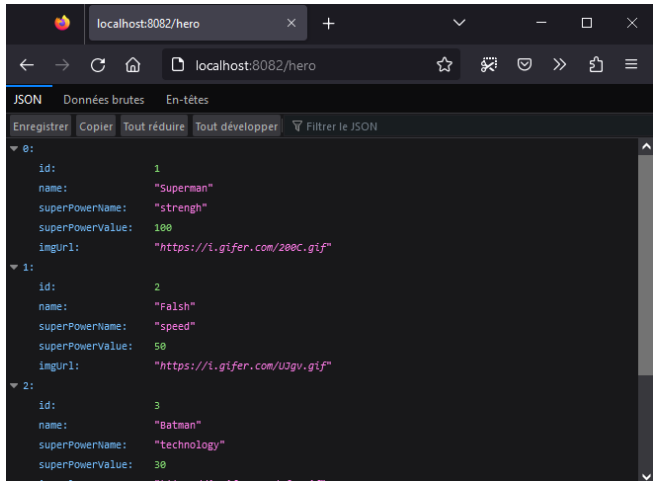
Spring Security

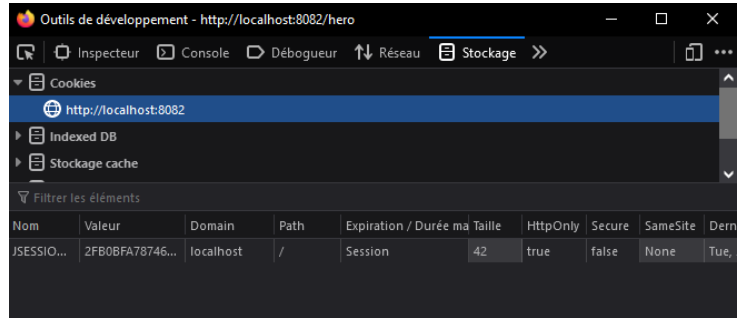












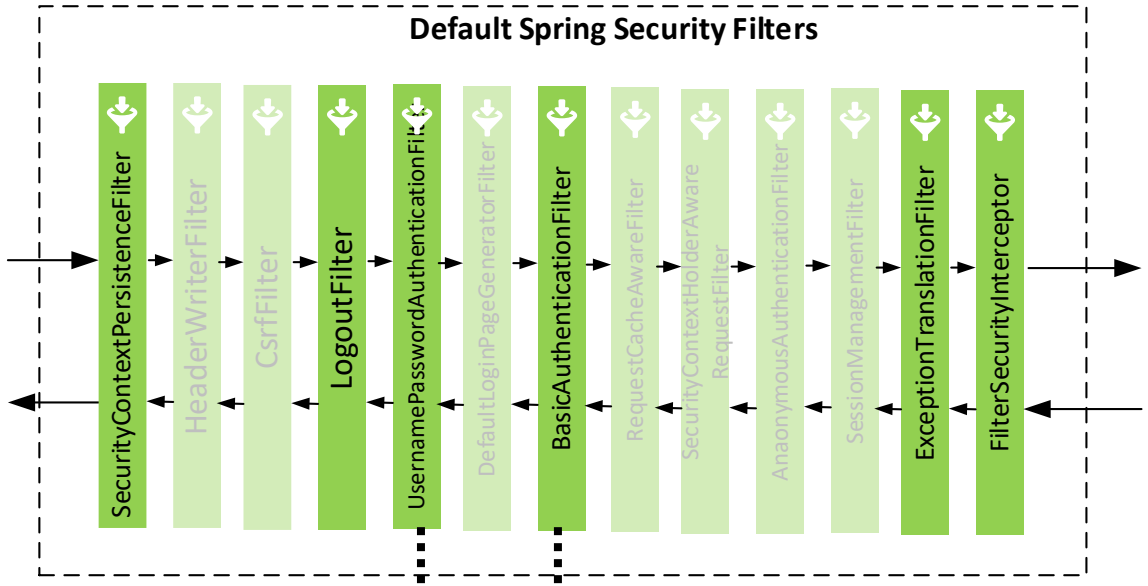

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.8)

2023-01-30 16:07:42.363  INFO 6432 --- [           main] com.security.app.HeroApp                 : Starting HeroApp using Java 11.0.8 on ARIA with
PID 6432 (D:\Data\cpe_cours\ASI\ASI-1\addDocs\ws\asi1-springboot-and-security\step-cours\target\classes started by jacques.saraydaryan in
D:\Data\cpe_cours\ASI\ASI-1\addDocs\ws\asi1-springboot-and-security\step-cours)
…
2023-01-30 16:07:45.396  INFO 6432 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation:
[org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-01-30 16:07:45.403  INFO 6432 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for
persistence unit 'default'
2023-01-30 16:07:45.786  WARN 6432 --- [           main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: 3f298f4d-1a86-4b55-8b28-3191024097b6

This generated password is for development use only. Your security configuration must be updated
before running your application in production.

2023-01-30 16:07:45.941  INFO 6432 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with …
2023-01-30 16:07:45.999  WARN 6432 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default.
Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2023-01-30 16:07:46.411  INFO 6432 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8081 (http) with
context path ''
2023-01-30 16:07:46.420  INFO 6432 --- [           main] com.security.app.HeroApp                 : Started HeroApp in 4.434 seconds (JVM running for
4.889)
```

A vous de Jouer !

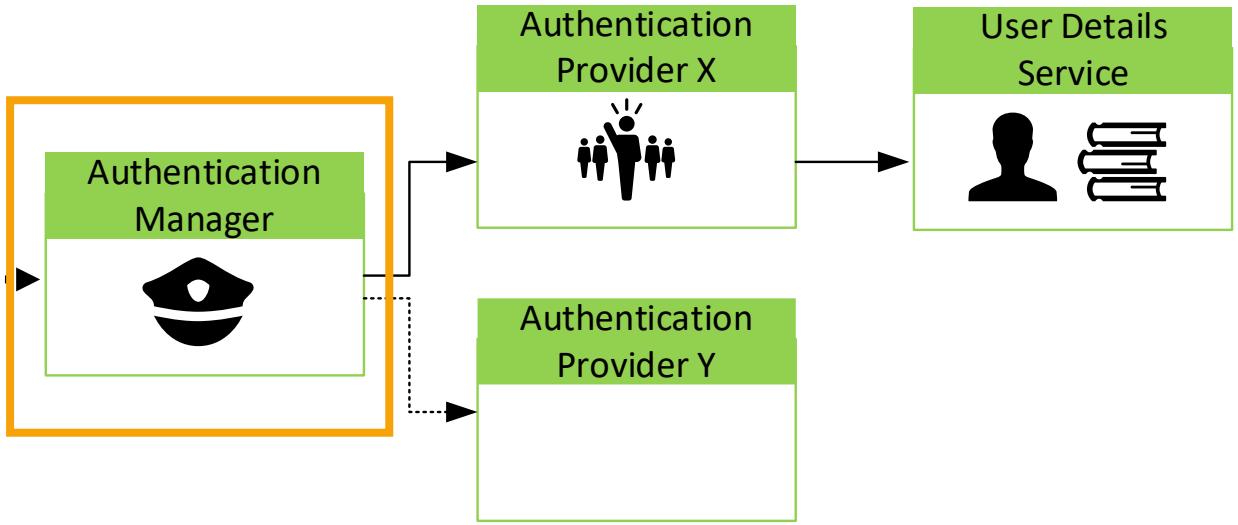
- ❑ Utilisation des Filtres et Intercepteurs
  - Step1
- ❑ Mise en place d'une première configuration
  - Step 2

https://gitlab.com/js-as1/asi1-springboot-and-security

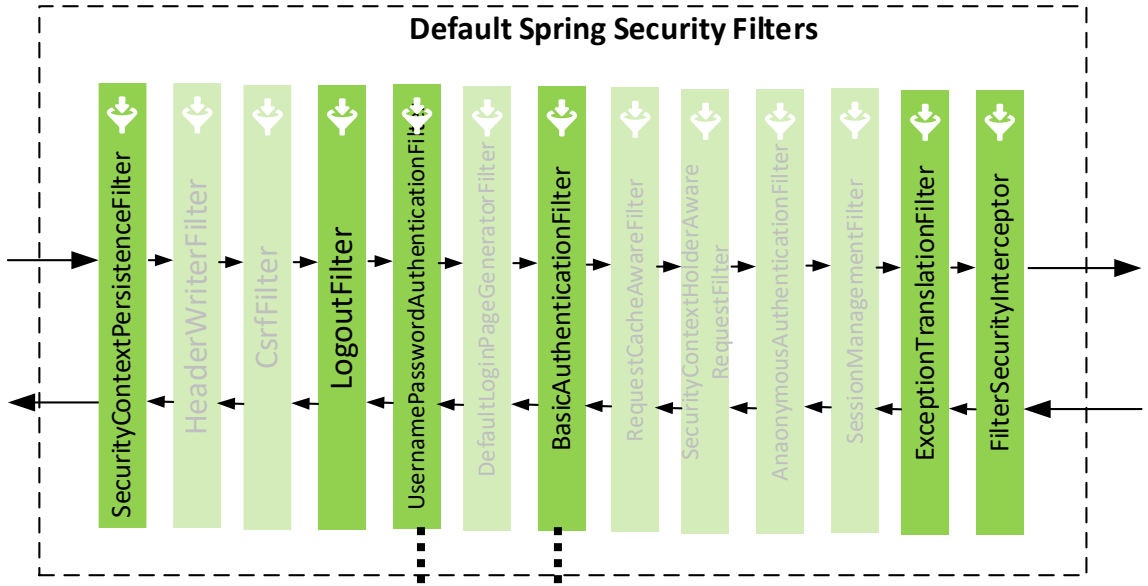
Authentification



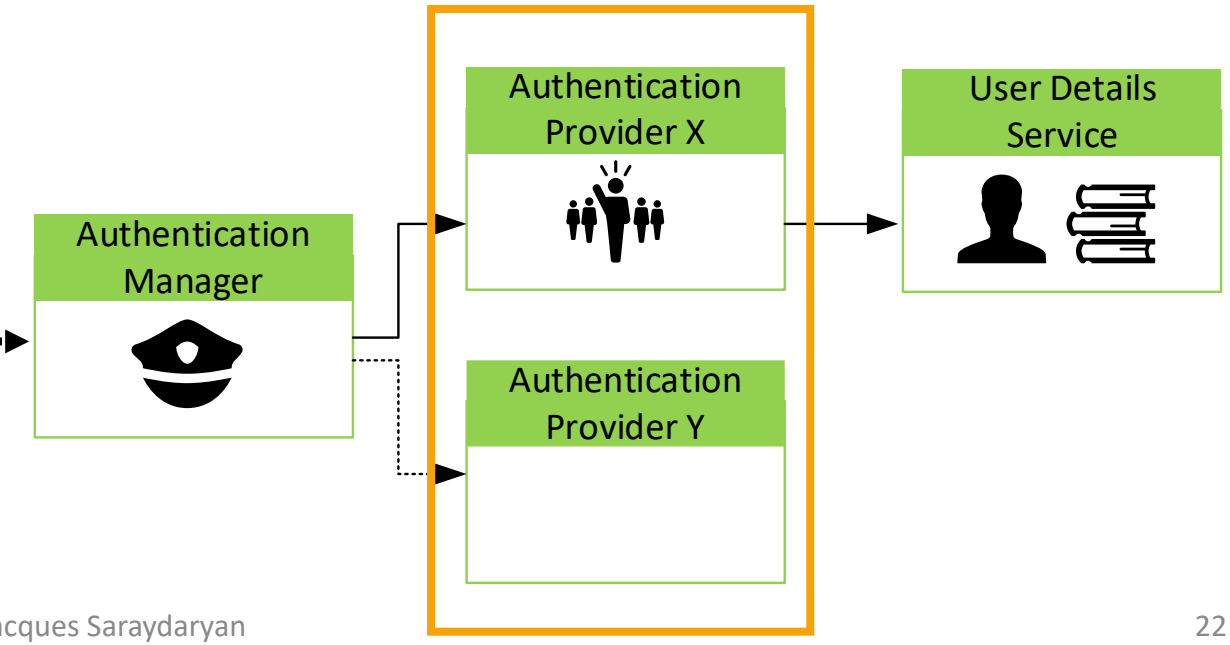
- AuthenticationManager
  - Api définissant comment l'authentification sera réalisée. L'implémentation la plus répandue est **`ProviderManager`** qui délègue l'authentification à une liste de **`AuthenticationProvider`**



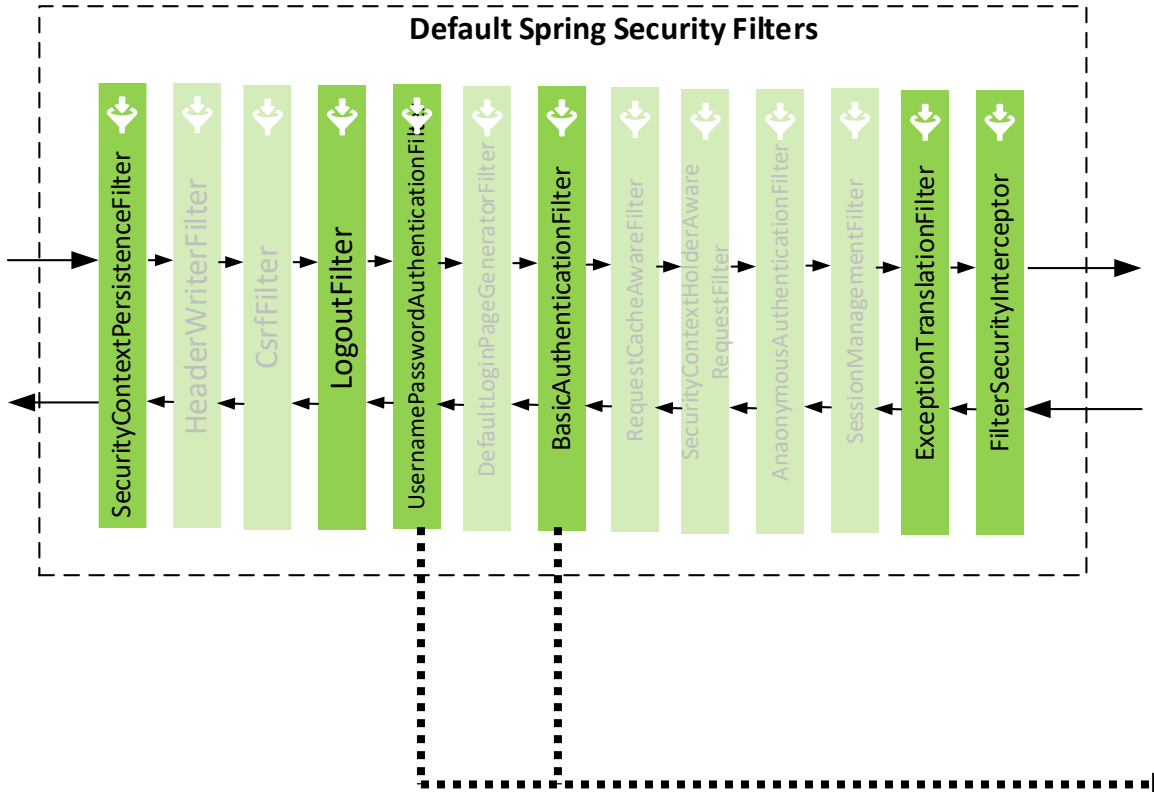
Authentification



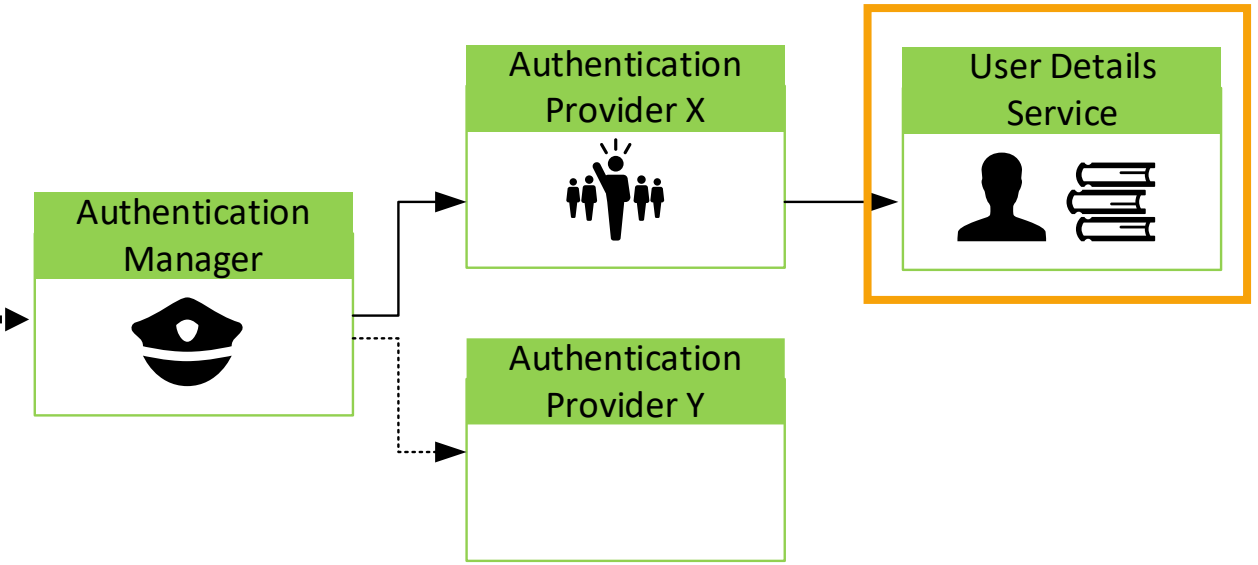
- AuthenticationProvider
  - Gère/supporte (e.g ldap, dao, JWT, Oauth…) un type d'authentification et décide si l'auth. est success ou failed ou indique si il est en mesure de prendre une décision ou non



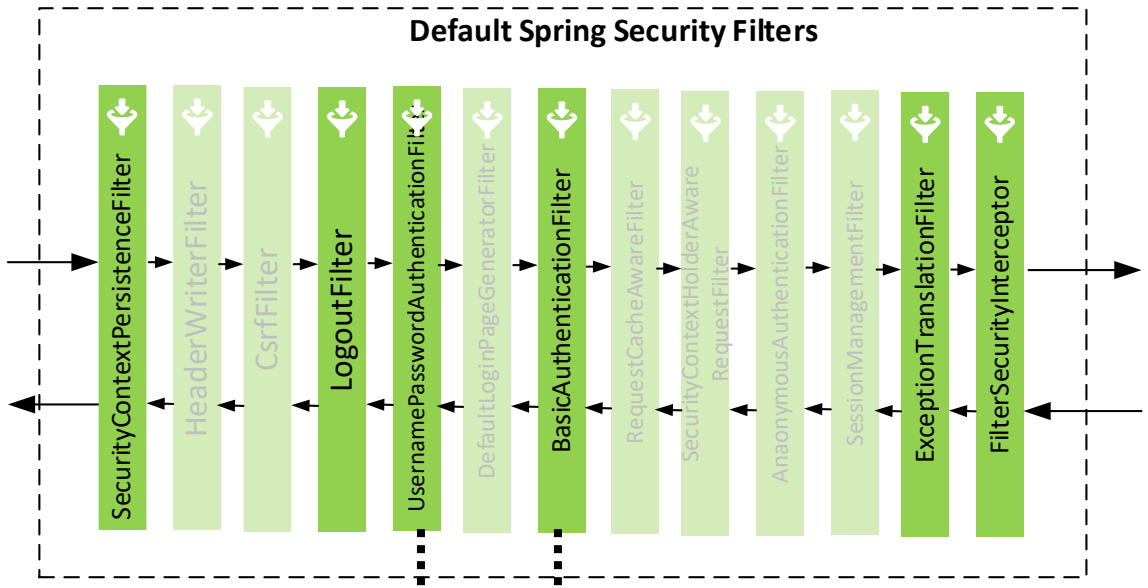
Authentification



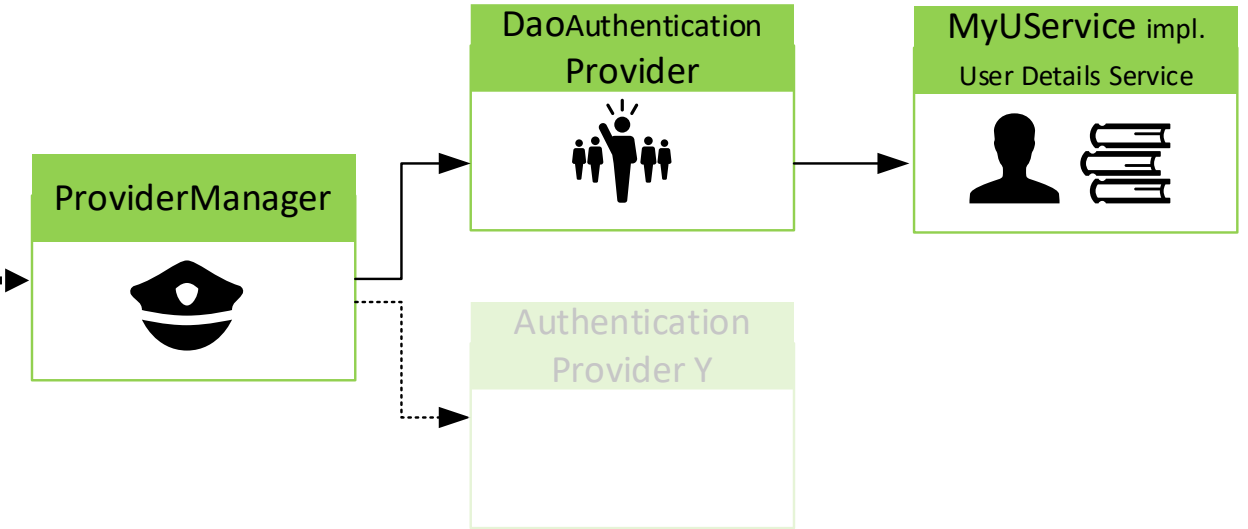
- `UserDetailsService`
  - Utilisé par certains **`AuthenticationProvider`** (e.g `DaoAuthenticationProvider`) pour récupérer les éléments nécessaires pour l'authentification d'un utilisateur (e.g username, password,…)



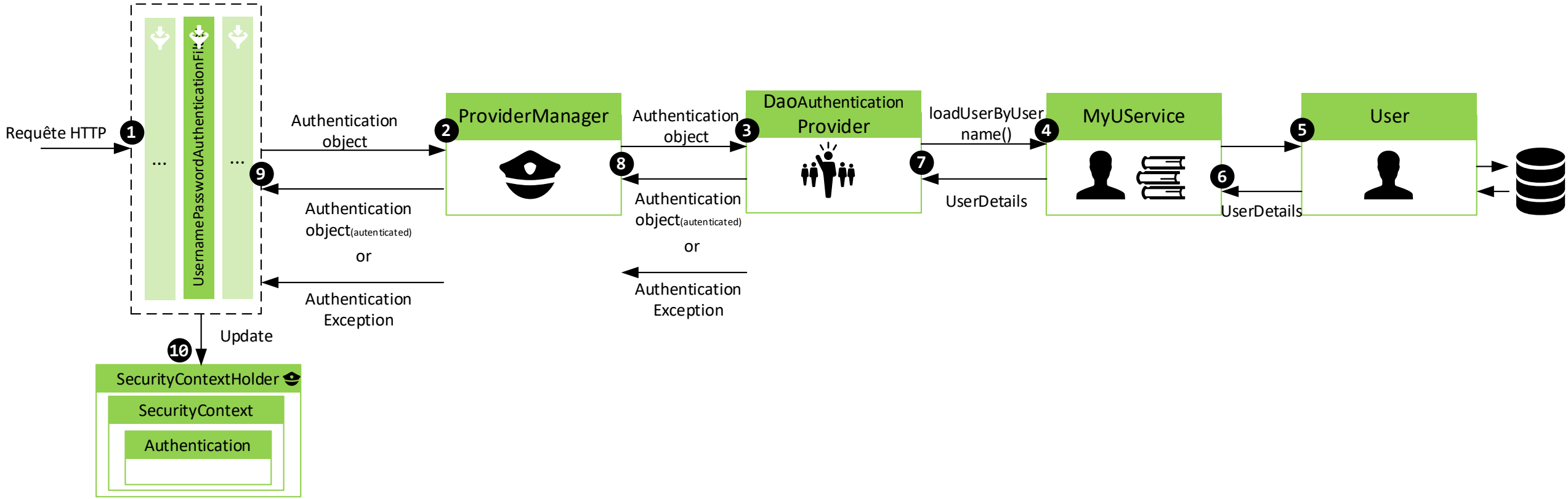
Authentification



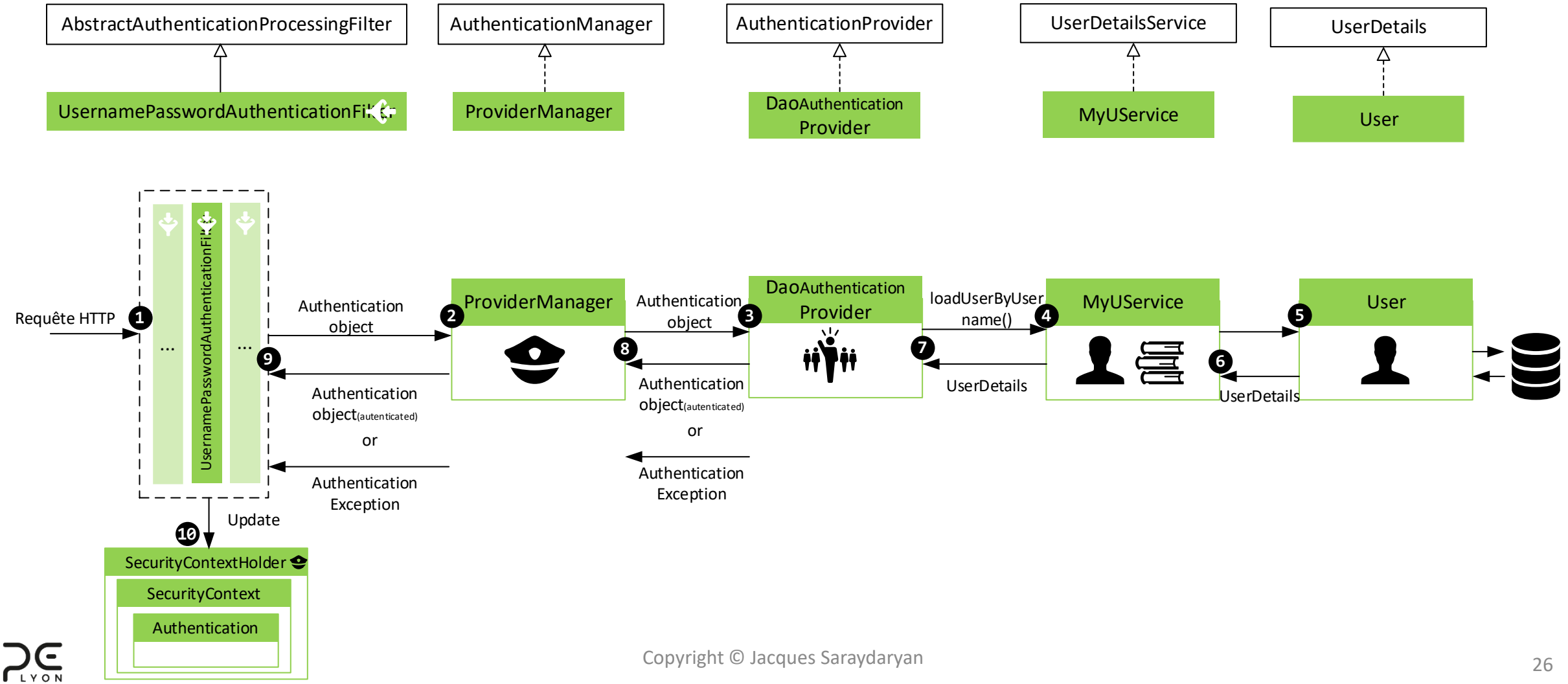
- ❑ **UsernamePasswordAuthenticationFilter**
- ❑ **ProviderManager**
- ❑ **DaoAuthenticationProvider**
- ❑ Service étendant **UserDetailsService**



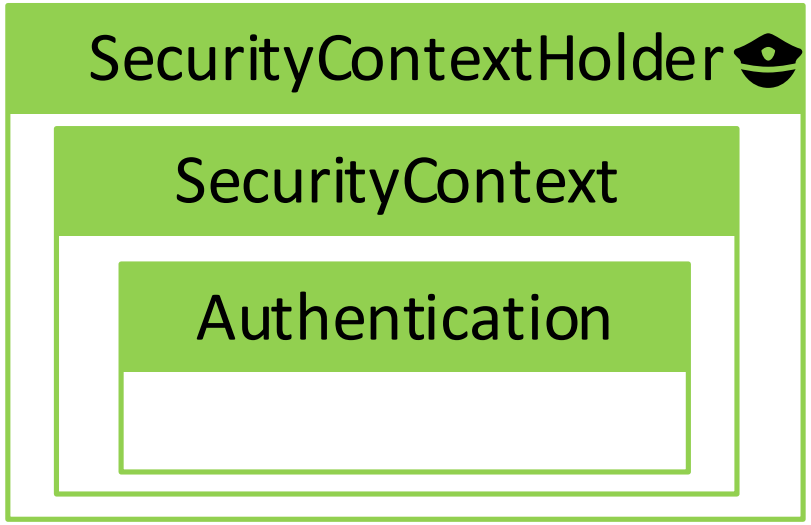
Authentification Flow



Authentification Flow



Authentification Flow

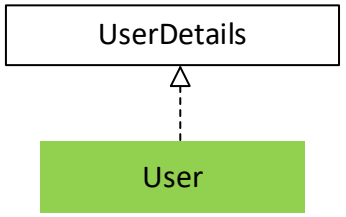


- `SecurityContextHolder`
  - Central pour le modèle d'authentification de Spring Security
  - Contient le `SecurityContext` qui contient des détails sur l'utilisateur authentifié courant
  - `SecurityContext` contient un object `Authentication`

Authentification

- Création d'un User
  - Implements UserDetails
  - Sucharge les méthodes indispensables

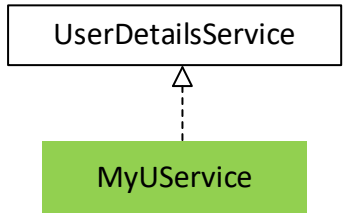
```
- getPassword()
- getUsername()
- isAccountNonExpired();
- isAccountNonLocked();
- isCredentialsNonExpired();
- …
```




```
Import …;

@Entity
@Table(name = "APPUSER")
public class User implements Serializable,UserDetails{
    private static final long serialVersionUID = 4668602180892212165L;

     @Id
     @GeneratedValue(strategy = GenerationType.IDENTITY)
     private Integer userId;
     private String username;
     private String password;

       //Getter and Setter
     @Override
      public Collection<? extends GrantedAuthority> getAuthorities()
     {
         return null;
     }
     @Override
     public boolean isAccountNonExpired() {
         return false;
     }
      @Override
     public boolean isAccountNonLocked() {
         return false;
     }
…
```

Authentification

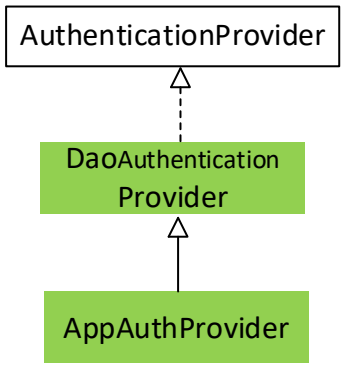
- Création d'un `MyUService`
  - extends `UserDetailsService`
  - Sucharge les méthodes indispensables
    - `loadUserByUsername(String username)`




```java
import …

@Service
public class MyUService implements UserDetailsService{

    private final UserRepository uRepo;

    public MyUService(UserRepository uRepo) {
        this.uRepo=uRepo;
    }

    @Override
    public UserDetails loadUserByUsername(String username)
                                        throws UsernameNotFoundException {
          User user = uRepo.findUserByUsername(username)
                    .orElseThrow(
                            () -> new UsernameNotFoundException(
                                                    "User not found"));
        return user;
    }

}
```

Authentification

- Création d'un AppAuthProvider
  - extends DaoAuthenticationProvider
  - Sucharge les méthodes indispensables
    - authenticate(Authentication authentication)
    - supports(Class<?> authentication)
    - …




```java
import …;
public class AppAuthProvider extends DaoAuthenticationProvider{

    private final MyUService uService;

    public AppAuthProvider(MyUService uService) {
        this.uService = uService;
    }

    @Override
    public Authentication authenticate(Authentication authentication)
                            throws AuthenticationException {
        UsernamePasswordAuthenticationToken auth =
                    (UsernamePasswordAuthenticationToken) authentication;

        String name = auth.getName();
        String password = auth.getCredentials().toString();
        UserDetails user = uService.loadUserByUsername(name);

        if (user == null) {
           throw new BadCredentialsException("Bad Auth"+
                                                    auth.getPrincipal());

        }else if(! user.getPassword().equals(password)) {
           throw new BadCredentialsException("Bad Auth" +
                                                    auth.getPrincipal());
        }
        return new UsernamePasswordAuthenticationToken(user, null,
                                                  user.getAuthorities());
    }
        @Override
    public boolean supports(Class<?> authentication) {
    return authentication.equals(
                            UsernamePasswordAuthenticationToken.class);}}
```

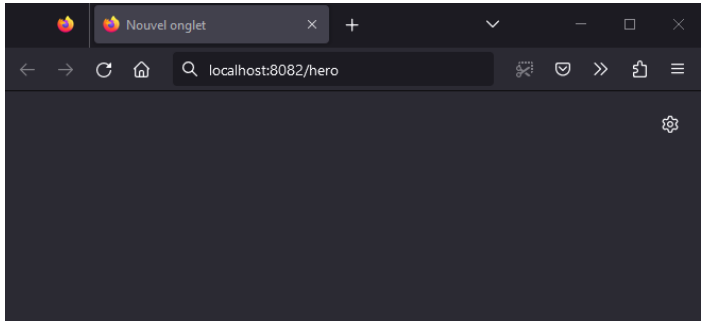
Authentification

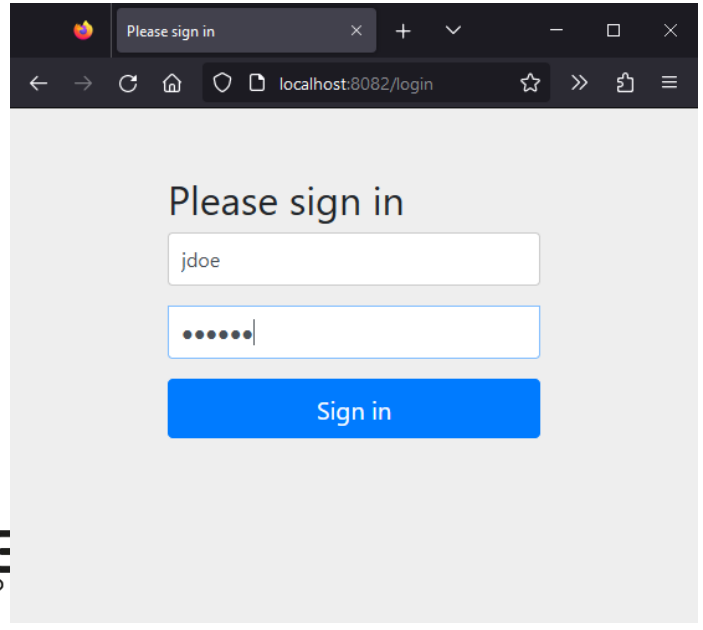
- Associe l'AuthProvider à Spring Security
  - Mise à jour de la configuration de sécurité
  - ProviderManager est créé par default
  - Le bean sera en charge de fournir l'instance de l'AuthentificationProvider définie (AppAuthProvider)

```
@Configuration
@EnableWebSecurity

public class SecurityConfig {

      …
@Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
            …
        http.authenticationProvider(getProvider());
            …
            }


    @Bean
    public AuthenticationProvider getProvider() {

        AppAuthProvider provider = new AppAuthProvider(uService);
        provider.setUserDetailsService(uService);

        return provider;
    }

}
```

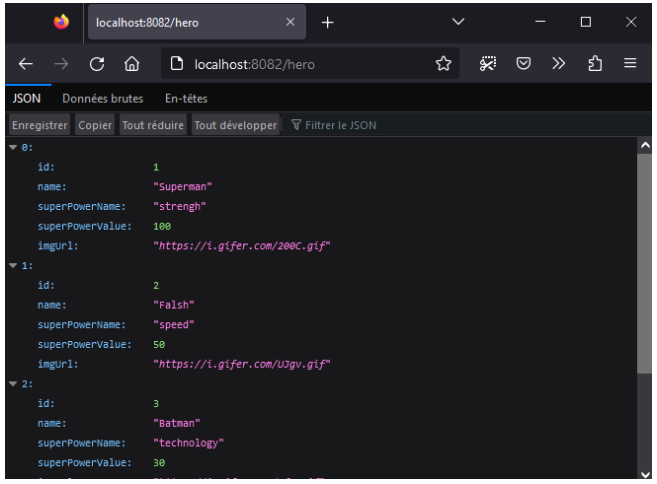
Authentification

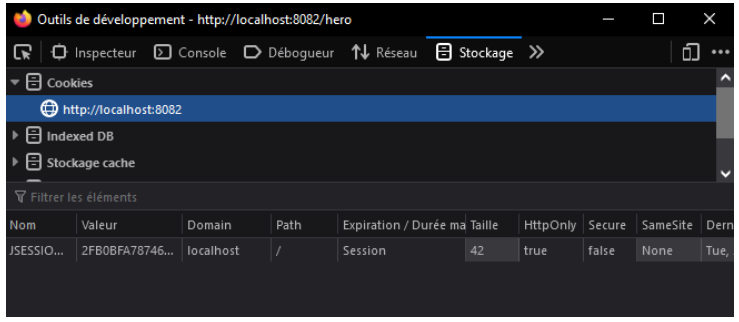
Data.sql






```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.8)

2023-01-31 11:59:44.828  INFO 8036 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will
secure any request with [org.springframework.security.web.session.DisableEncodeUrlFilter@50d4775b,
org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@642614b7,
org.springframework.security.web.context.SecurityContextPersistenceFilter@71d2261e,
org.springframework.security.web.header.HeaderWriterFilter@497fd334,
org.springframework.security.web.authentication.logout.LogoutFilter@73baf7f0,
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@33a7331,
org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter@27d6467,
org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@345540f9,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@1fa796a4,
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@68de8522,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@37344d31,
org.springframework.security.web.session.SessionManagementFilter@6f89ad03,
org.springframework.security.web.access.ExceptionTranslationFilter@14ba7f15,
org.springframework.security.web.access.intercept.FilterSecurityInterceptor@3e18e693] 2023-01-31 11:59:45.353
INFO 8036 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8082 (http) with context path ''
2023-01-31 11:59:45.362  INFO 8036 --- [           main] com.security.app.HeroApp                 : Started
HeroApp in 4.496 seconds (JVM running for 4.851)
```



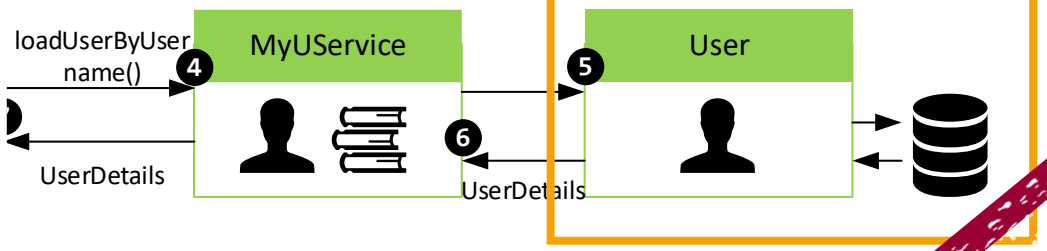


A vous de Jouer !

- Mise en place d'une authentification (Session)
  - Step3

https://gitlab.com/js-as1/asi1-springboot-and-security

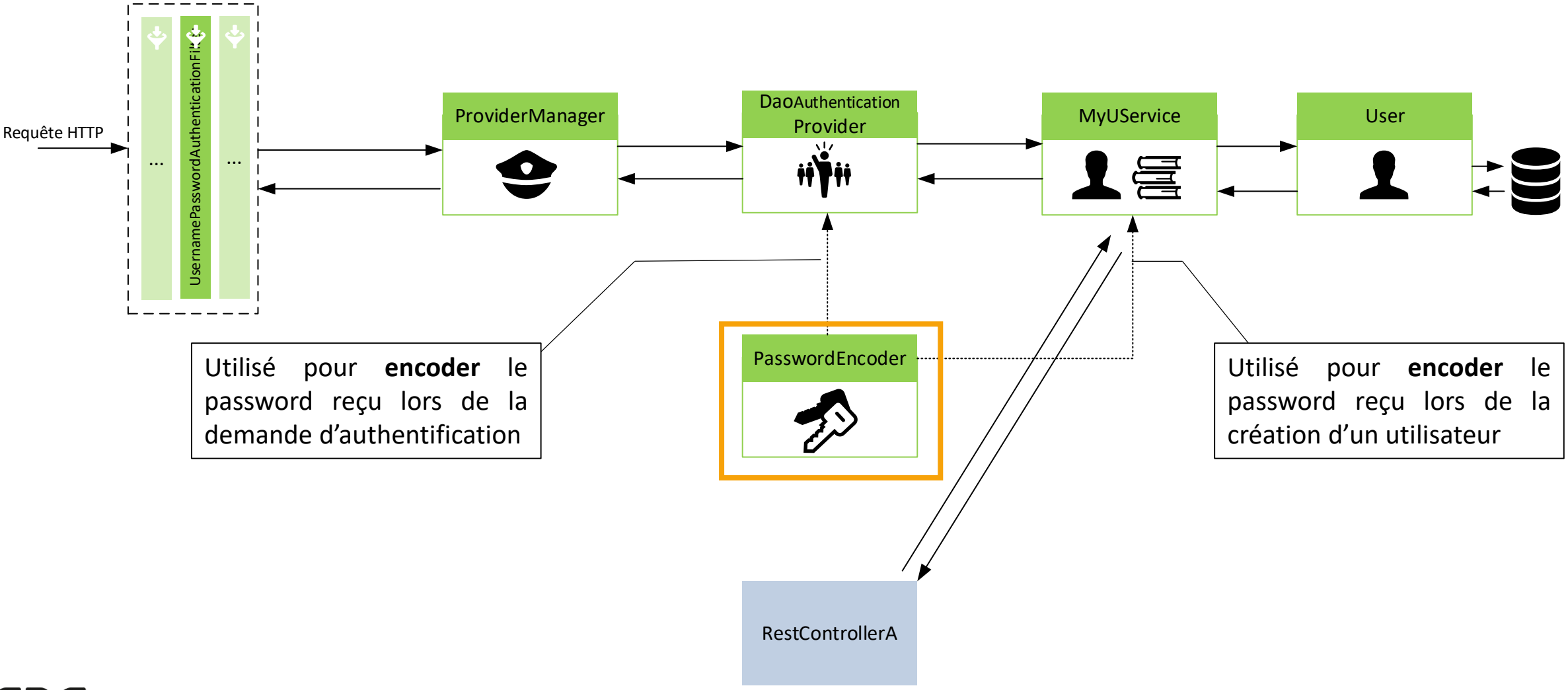
Stockage du password




```
@Entity
@Table(name = "APPUSER")
public class User implements Serializable,UserDetails{
    private static final long serialVersionUID = 4668602180892212165L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer userId;
    private String username;
    private String password;

    @Override
    public String getPassword() {
        return password;
    }
```

Password Stocké en clair !

Stockage du password



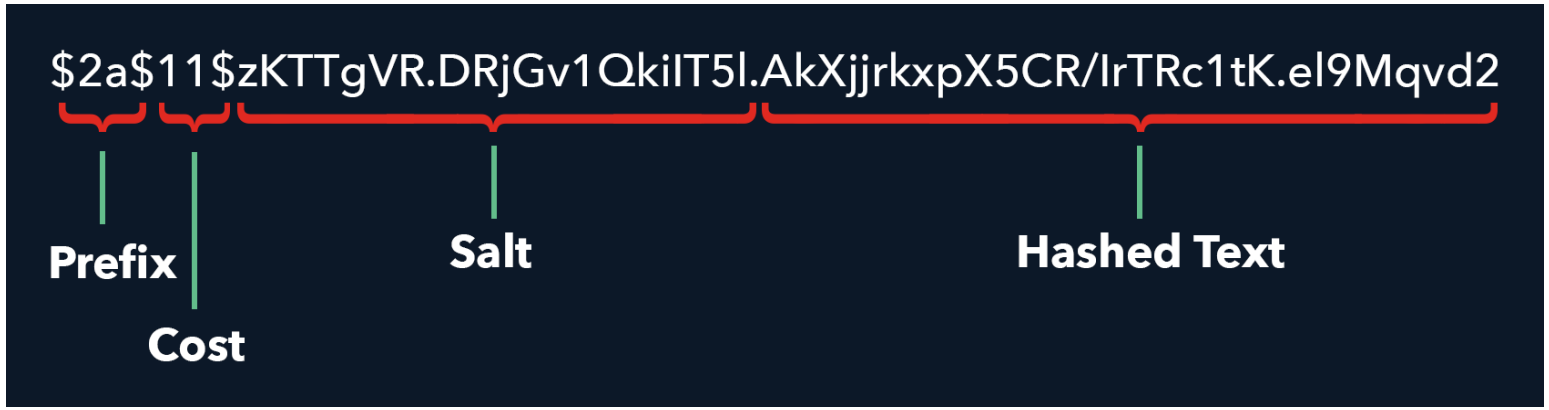
Stockage de password

- Définition d'un service de `PasswordEncoder`
- Usage d'un utilitaire `Bcrypt` pour encoder les passwords

```java
package com.security.app.config.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;



@Configuration
public class MyBCryptPasswordEncoder {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Bcrypt

- Niels Provos, David Mazières 1999
- Stockage de mots de passe (usage principal)
- Usage de grain de sel (protection contre Rainbow tables)
- Fonction adaptative (augmentation du nombre d'itérations possible)
- Basé sur l'algorithme de chiffrement de Blowfish



Stockage de password

- Modification de `MyUservice`
- Utilisation du `PasswordEncoder` pour définir le password de l'utilisateur

```
package com.security.app.config.security;
import …;

@Service
public class MyUService implements UserDetailsService{
    private final UserRepository uRepo;
    public MyUService(UserRepository uRepo) {
        this.uRepo=uRepo;
    }

    @Autowired
    private finale PasswordEncoder passwordEncoder;

    public boolean addUser(UserDto userDto) {
        Optional<User> u =uRepo.findUserByUsername(userDto.getUsername());
        if( !u.isPresent()){
            User u_new=new User();
            u_new.setUsername(userDto.getUsername());
            u_new.setPassword(passwordEncoder.encode(userDto.getPassword())
);
            uRepo.save(u_new);
            return true;
        }
        return false;
```

Stockage de password

- Associe le passwordEncoder à l'AuthProvider

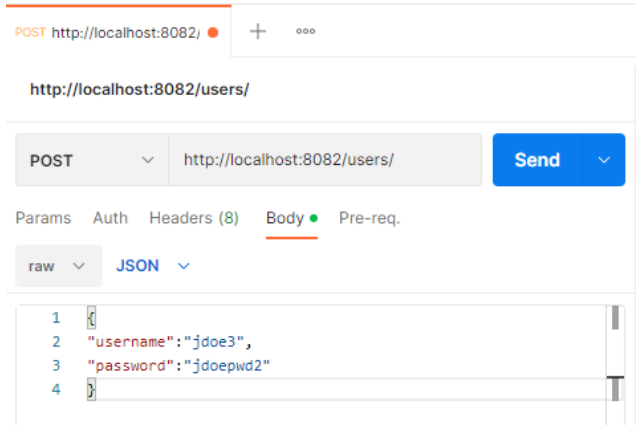
```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    private final MyUService uService;
    private final PasswordEncoder pEncoder;


    public SecurityConfig(MyUService uService, PasswordEncoder pEncoder) {
        this.uService=uService;
        this.pEncoder =pEncoder;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
                    …
    }

    @Bean
    public AuthenticationProvider getProvider() {
        AppAuthProvider provider = new AppAuthProvider(uService);
        provider.setUserDetailsService(uService);
        provider.setPasswordEncoder(passwordEncoder);
        return provider;
    }

}
```

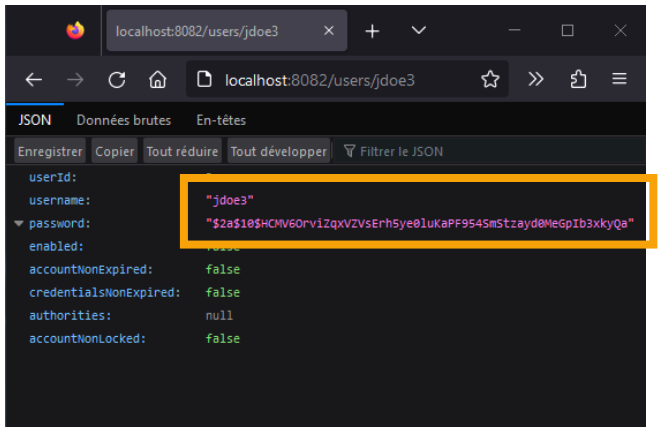
Stockage de password

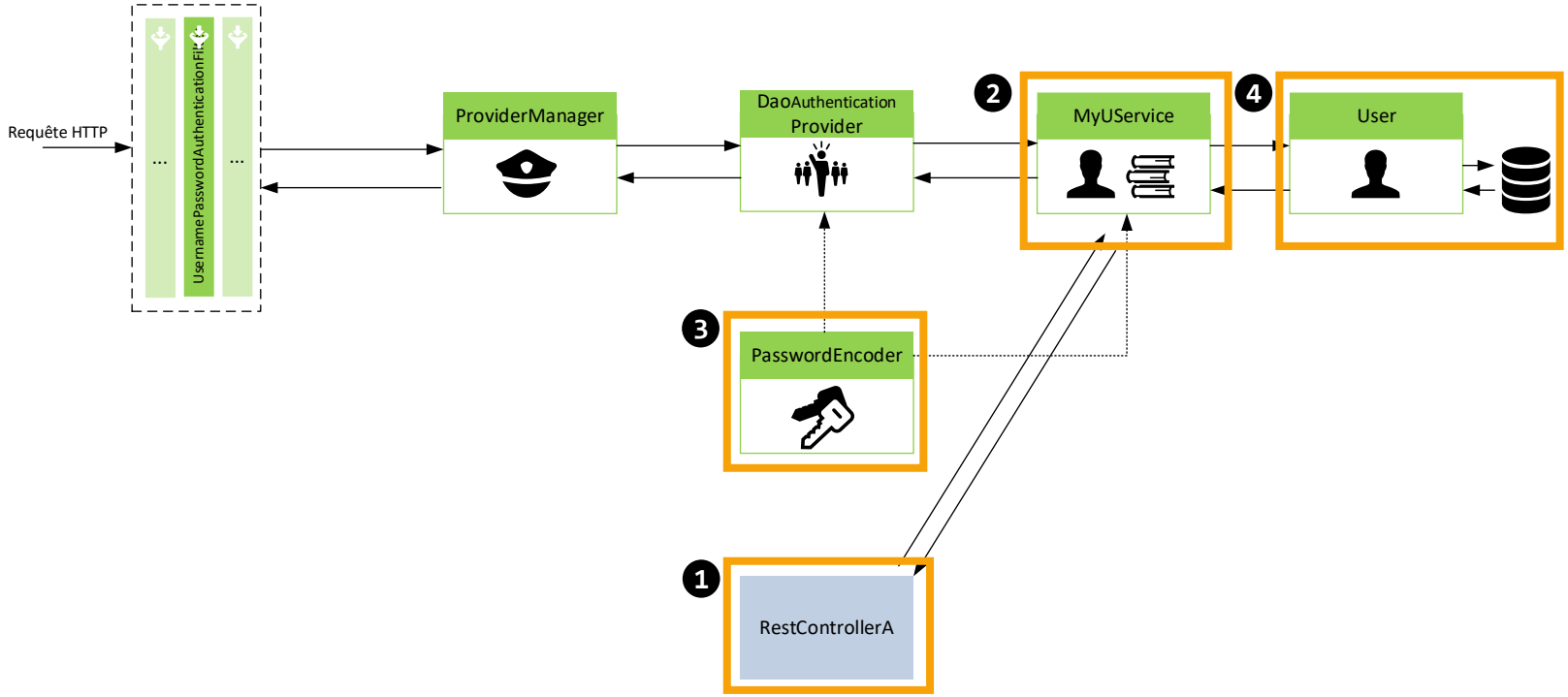
- Modification de AuthProvider custom (AppAuthProvider)

```
public class AppAuthProvider extends DaoAuthenticationProvider{

    private final MyUService uService;
    public AppAuthProvider(MyUService uService) {
        this.uService = uService;
    }

    @Override
    public Authentication authenticate(Authentication authentication)
                                            throws AuthenticationException {
        UsernamePasswordAuthenticationToken auth =
                        (UsernamePasswordAuthenticationToken) authentication;
        String name = auth.getName();
        String password_raw = auth.getCredentials().toString();
        UserDetails user = uService.loadUserByUsername(name);
        if (user == null) {
        throw new BadCredentialsException("Bad Auth"+ auth.getPrincipal());

        }else
            if(!this.getPasswordEncoder().matches(password_raw,
                                                    user.getPassword()))
            {
        throw new BadCredentialsException("Bad Auth"+auth.getPrincipal());
            }
        return new UsernamePasswordAuthenticationToken(user, null,
                                                    user.getAuthorities());
    }

      ...
}
```

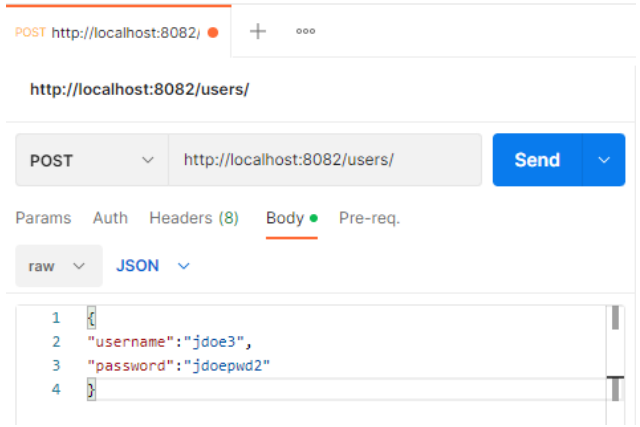
Stockage de password

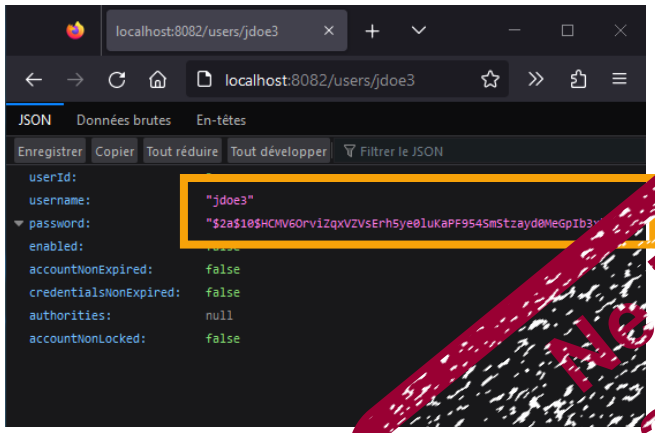


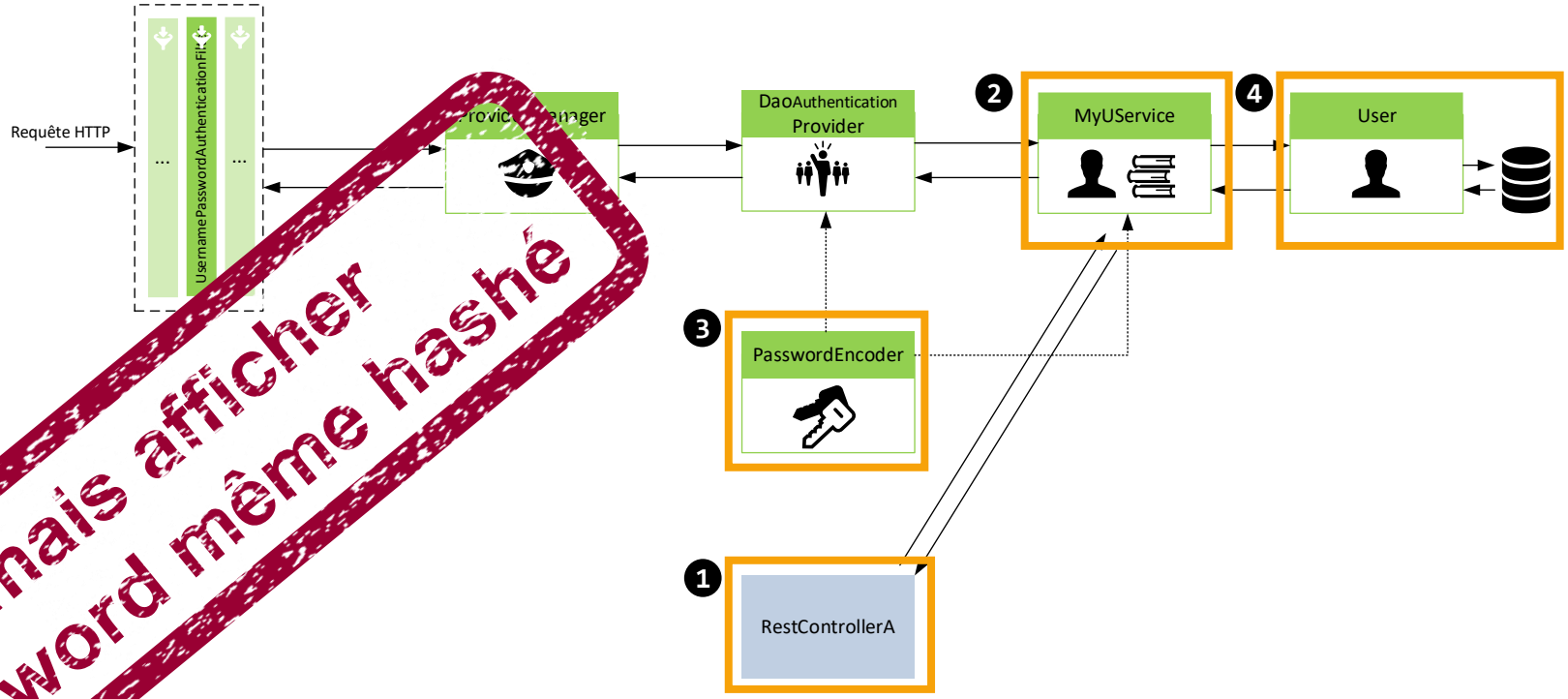




Stockage de password

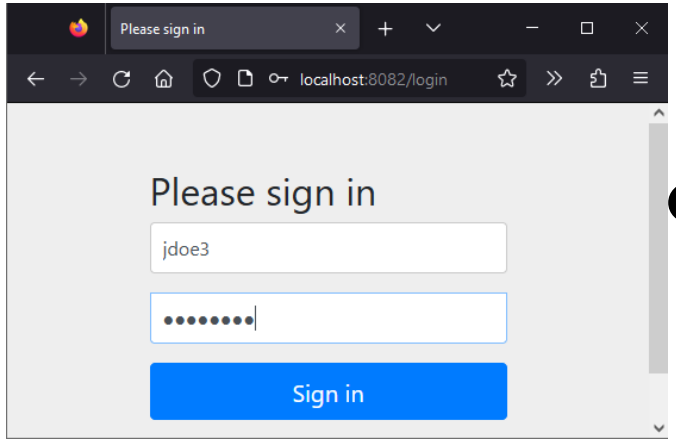


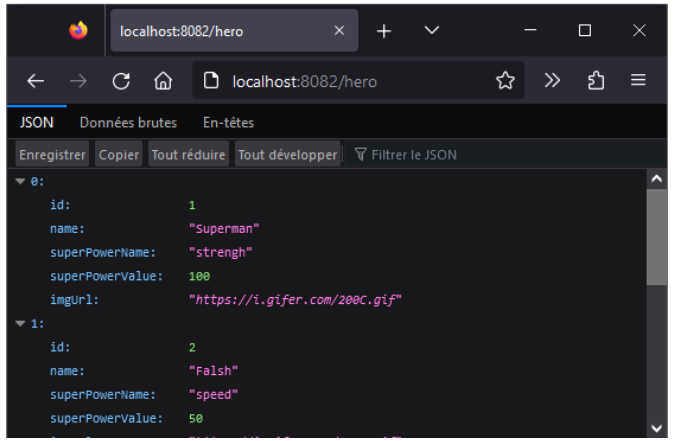


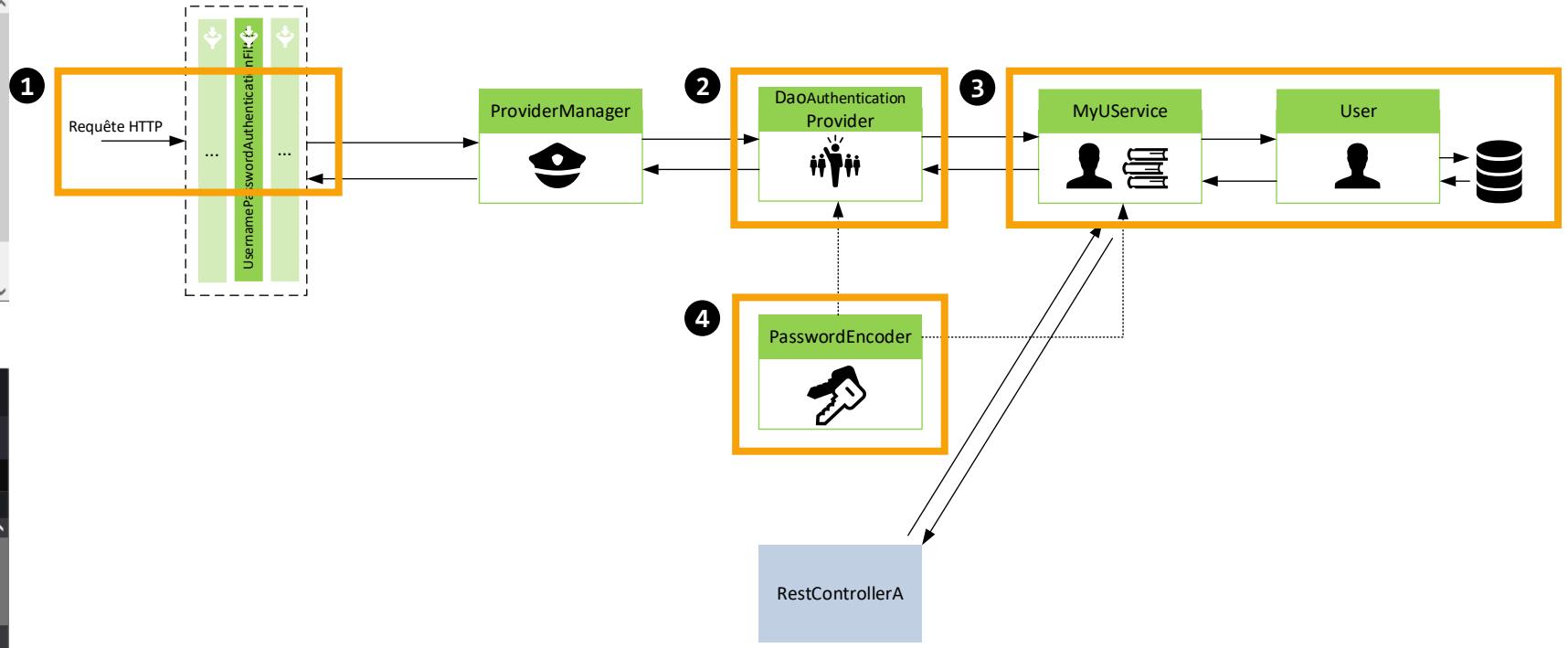


Ne jamais afficher
Le password même hashé

Spring Security





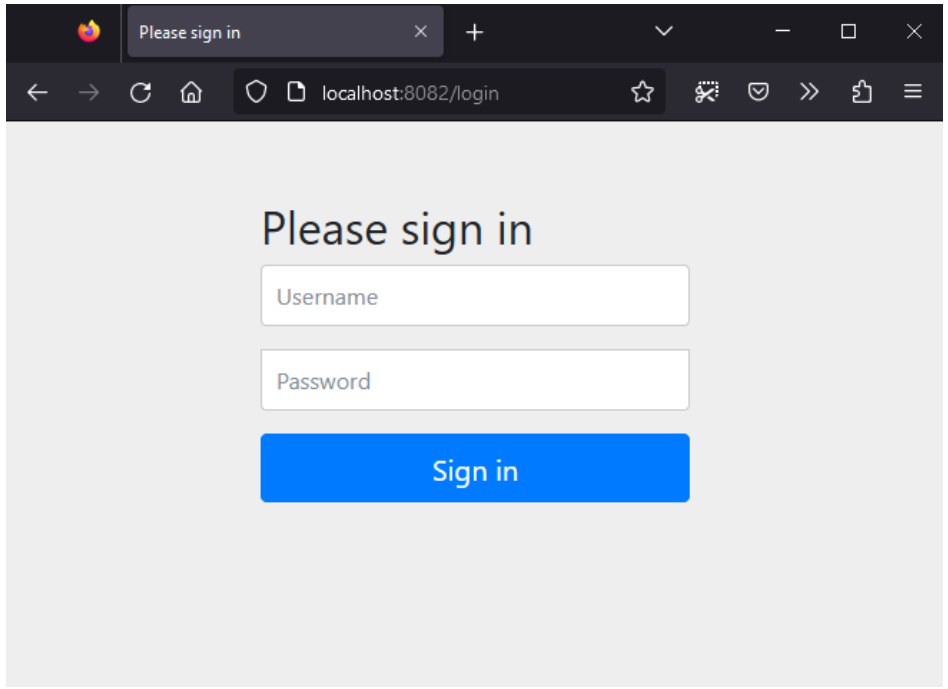


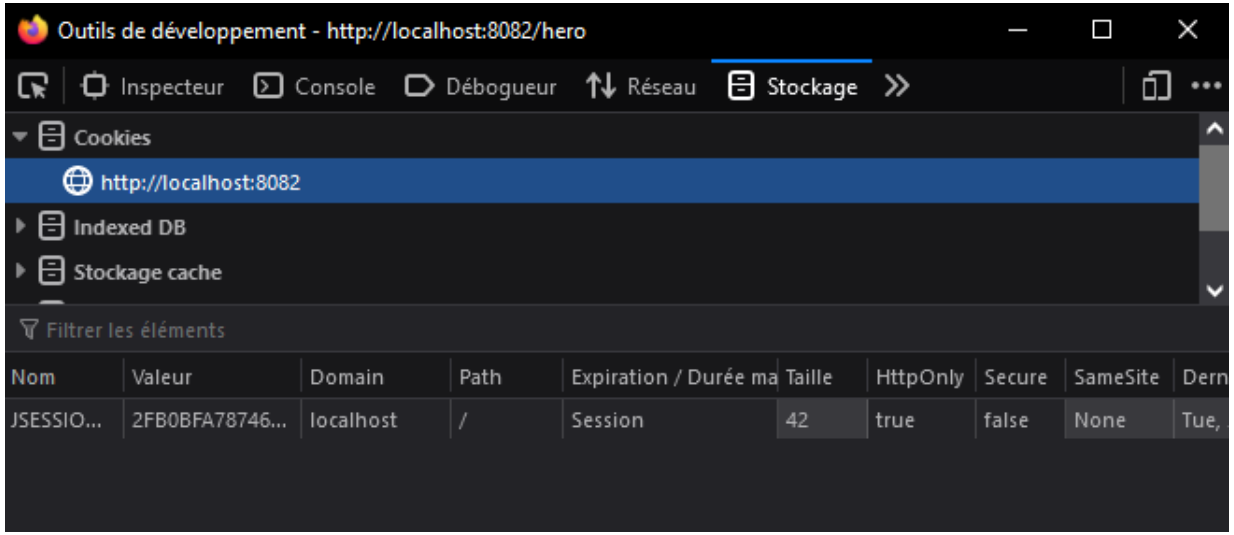
A vous de Jouer !

- Stockage de mot de passe sécurité
  - Step4

https://gitlab.com/js-as1/asi1-springboot-and-security

Authentification Statefull





Authentification

- Comment être compatible avec les architectures RESTFULL (stateless) ?

→ Utilisation de tokens signés (e.g JWT)

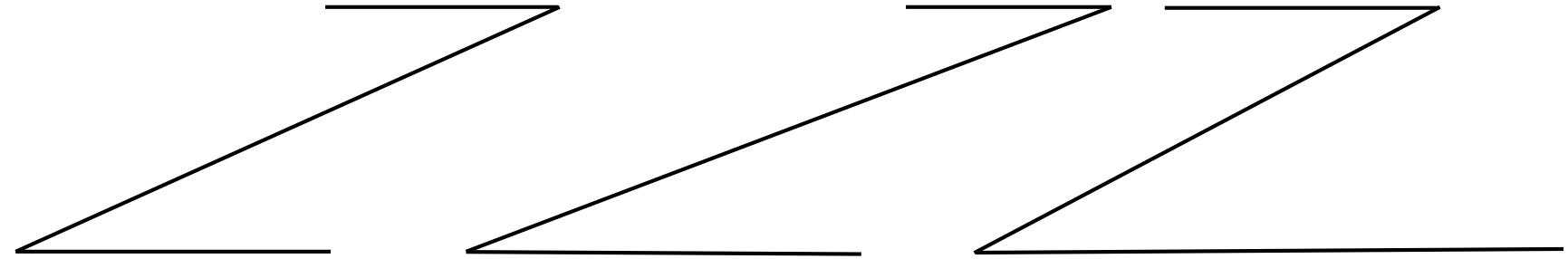
JSON Web Token (JWT)

- Header :
  - Type de token utilisé
  - Type d'algo utilisé pour la signature
- Payload
  - User defined attributes (public claims)
  - Some are standard (called reserved claims)
- JWT Signature (HMAC or RSA)
  - Header
  - Payload
  - Secret (hmac)

https://jwt.io/introduction/

JSON Web Token (JWT)

BASE64Url(**HEADER**).BASE64Url(**Payload**).**Signature**



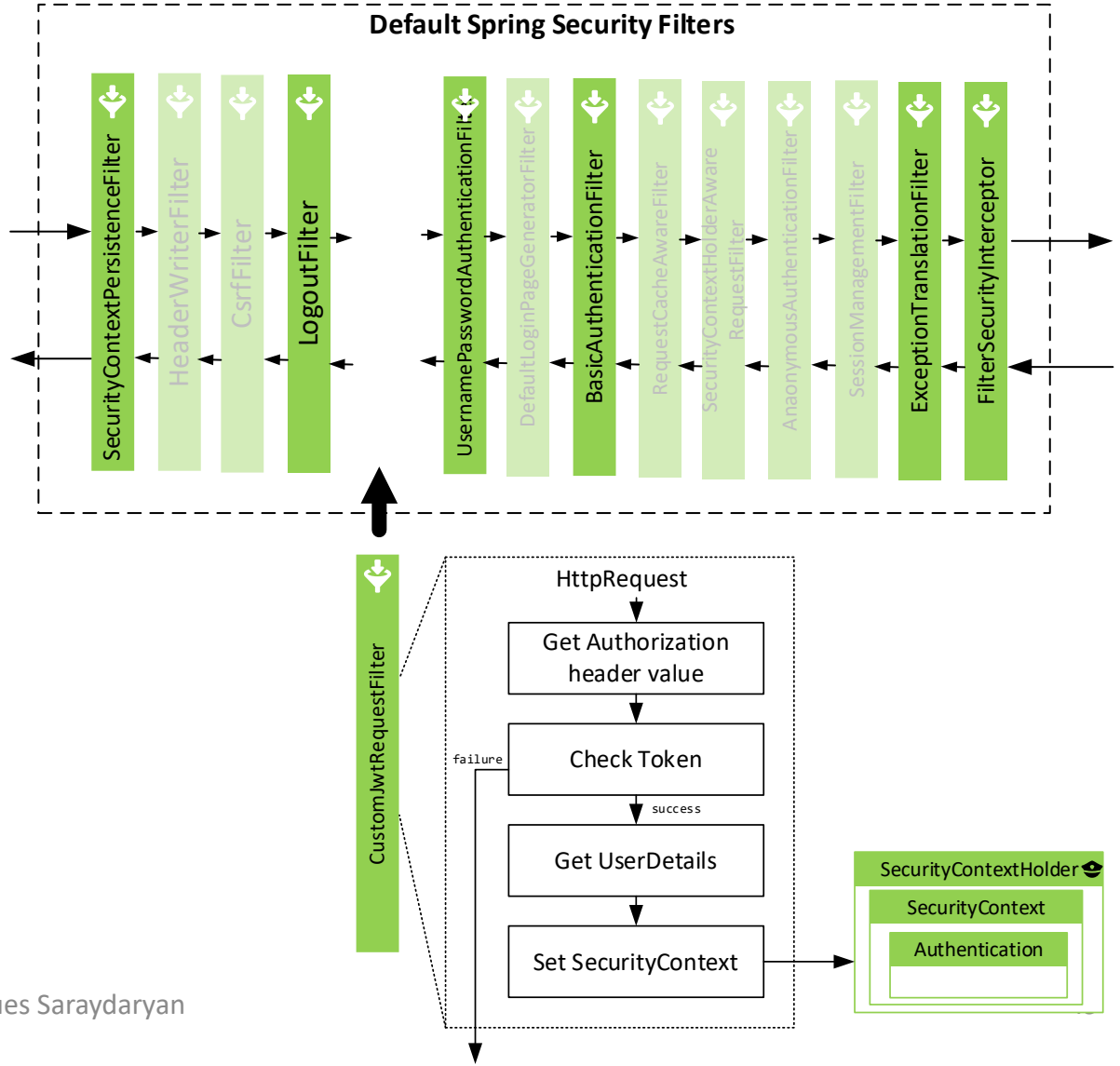
```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

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

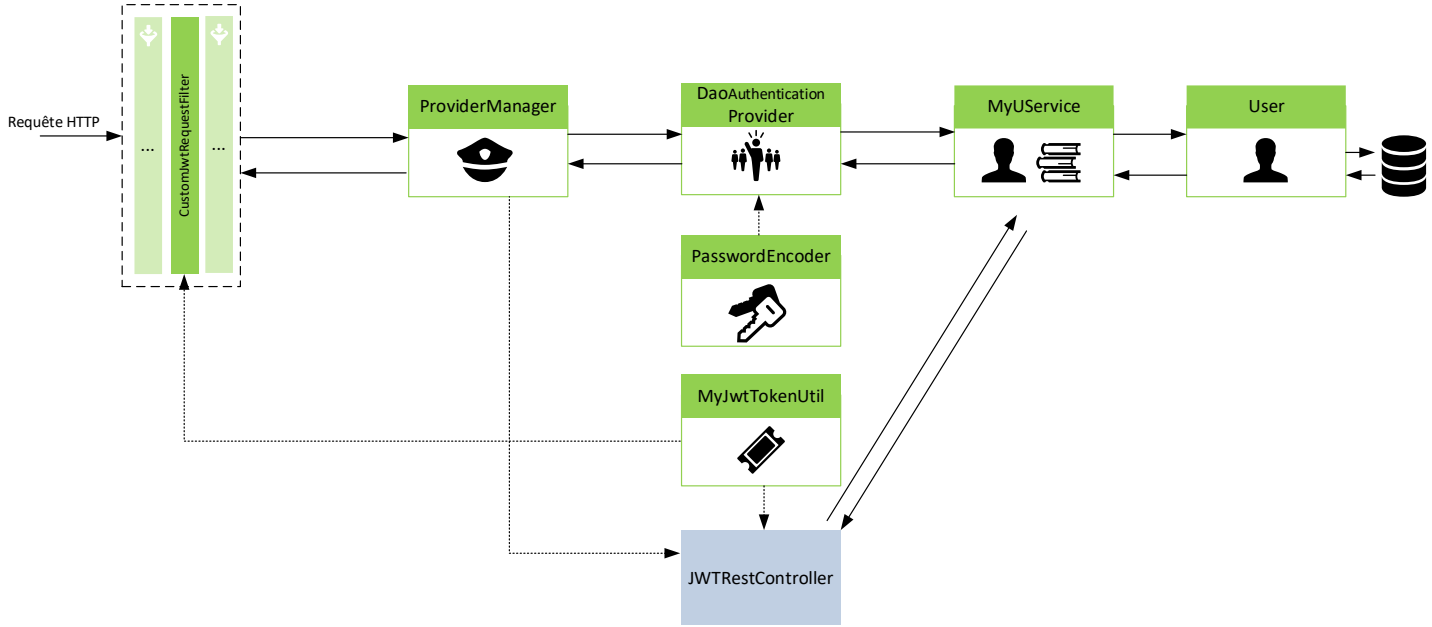
Authentification JWT (validation d'un token)

- Création d'un `Filter` permettant d'intercepter les requêtes
- Récupération et Vérification du token si il existe
- Ajout des informations au `SecurityContext` si le token est valide (il n'y aura pas d'authentification dans ce cas)



Authentification JWT (génération d'un token)

- Création d'un controller spécifique `JWTRestController` chargé de récupérer les demandes de connexion (e.g /login)
- Vérification du login/pwd
- Création d'un token si l'authentification réussit



Authentification JWT (gé

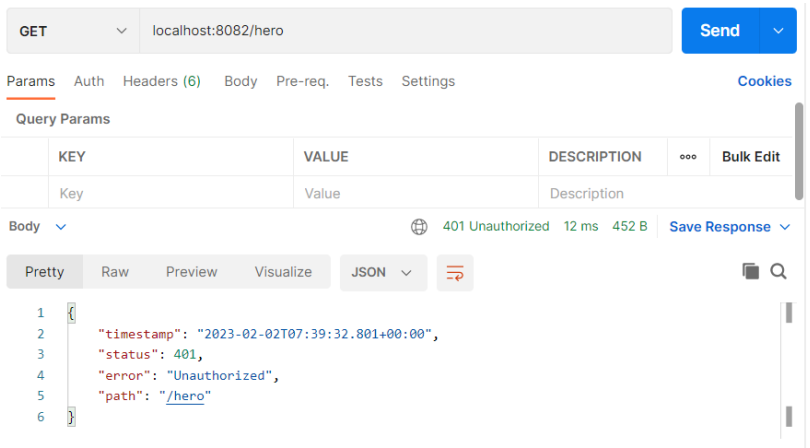
- Création d'un controller spécifique `JWTRestController` chargé de récupérer les demandes de connexion (e.g /login)
- Vérification du login/pwd
- Création d'un token si l'authentification réussit

```
@RestController
public class JwtAuthRestController {
    @Autowired
    private AuthenticationManager authenticationManager;
    @Autowired
    private JwtTokenUtil jwtTokenUtil;
    @Autowired
    AuthService authService;

    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public ResponseEntity<?> createAuthenticationToken(@RequestBody
                        JwtRequest authenticationRequest) throws Exception {
        authenticate(authenticationRequest.getUsername(),
                                            authenticationRequest.getPassword());
        final UserDetails userDetails =
      authService.loadUserByUsername(authenticationRequest.getUsername());
        final String token = jwtTokenUtil.generateToken(userDetails);
        return ResponseEntity.ok(new JwtResponse(token));
    }

    private void authenticate(String username, String password)
                                                            throws Exception {
        try {
            authenticationManager.authenticate(new
                    UsernamePasswordAuthenticationToken(username, password));
        } catch (DisabledException e) {
            throw new Exception("USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new Exception("INVALID_CREDENTIALS", e);
        }
    }
}
```

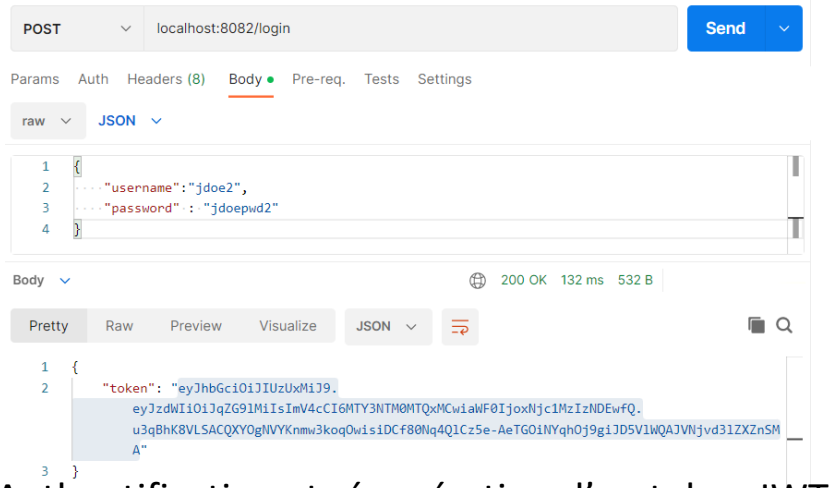
Authentification JWT (génération d'un token)

- TODO Ajouter JWTRequestFilter

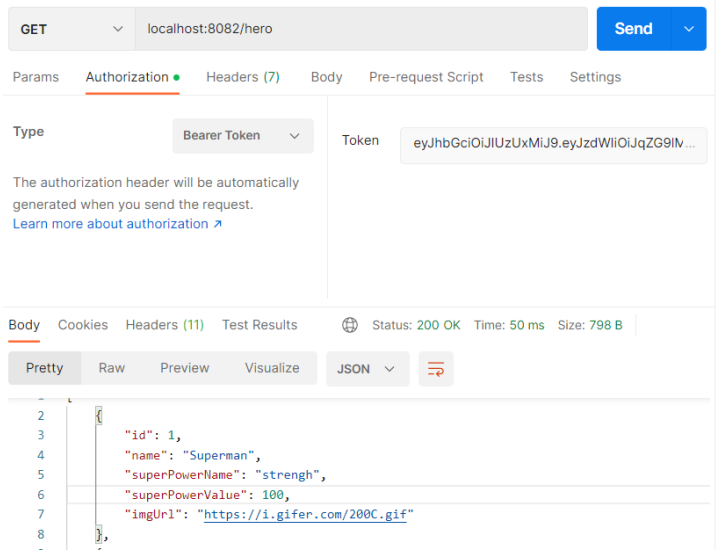
Authentification JWT



Accès non autorisé car pas authentifié (401)



Authentification et récupération d'un token JWT



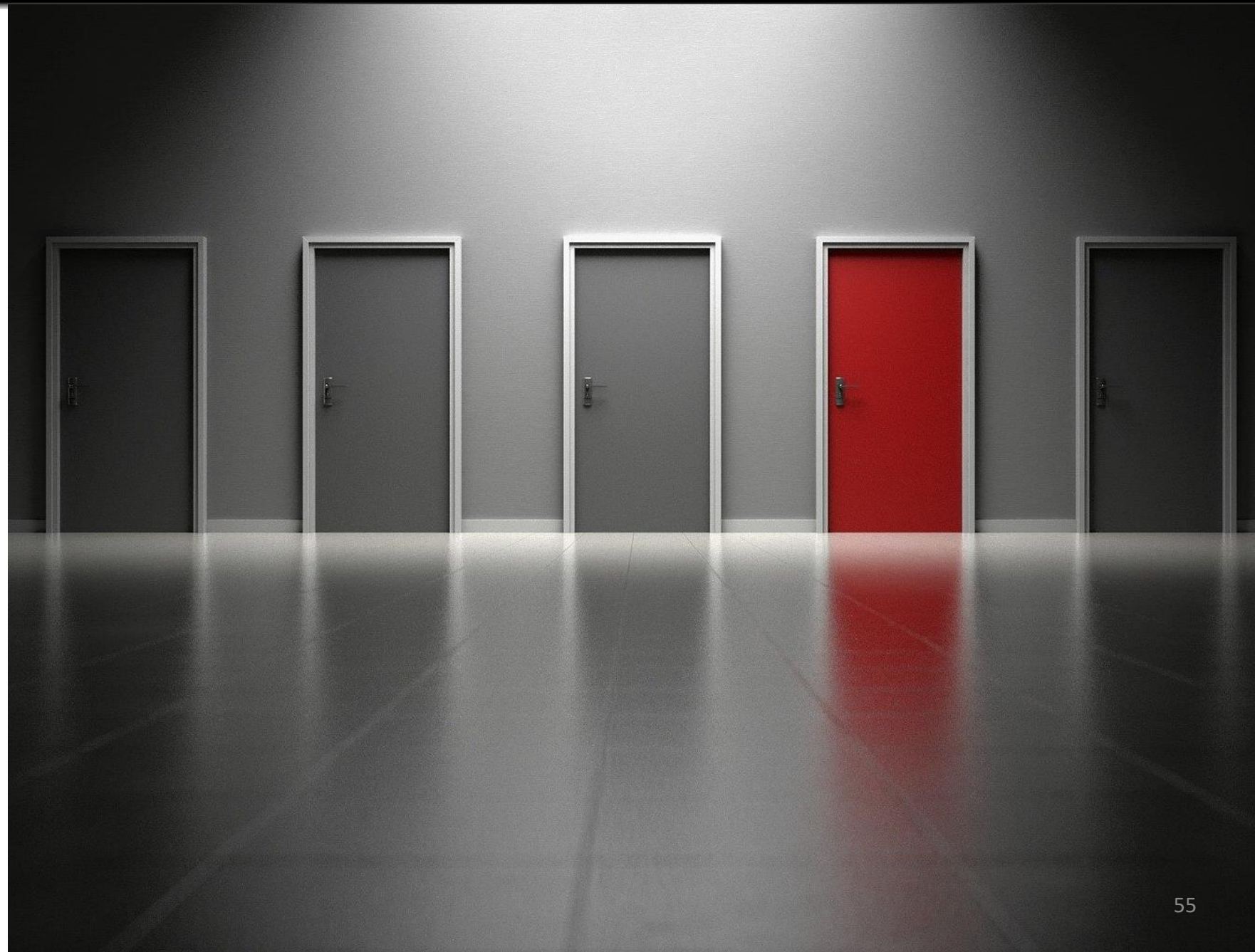
Envoie d'une requête avec le token JWT

A vous de Jouer !

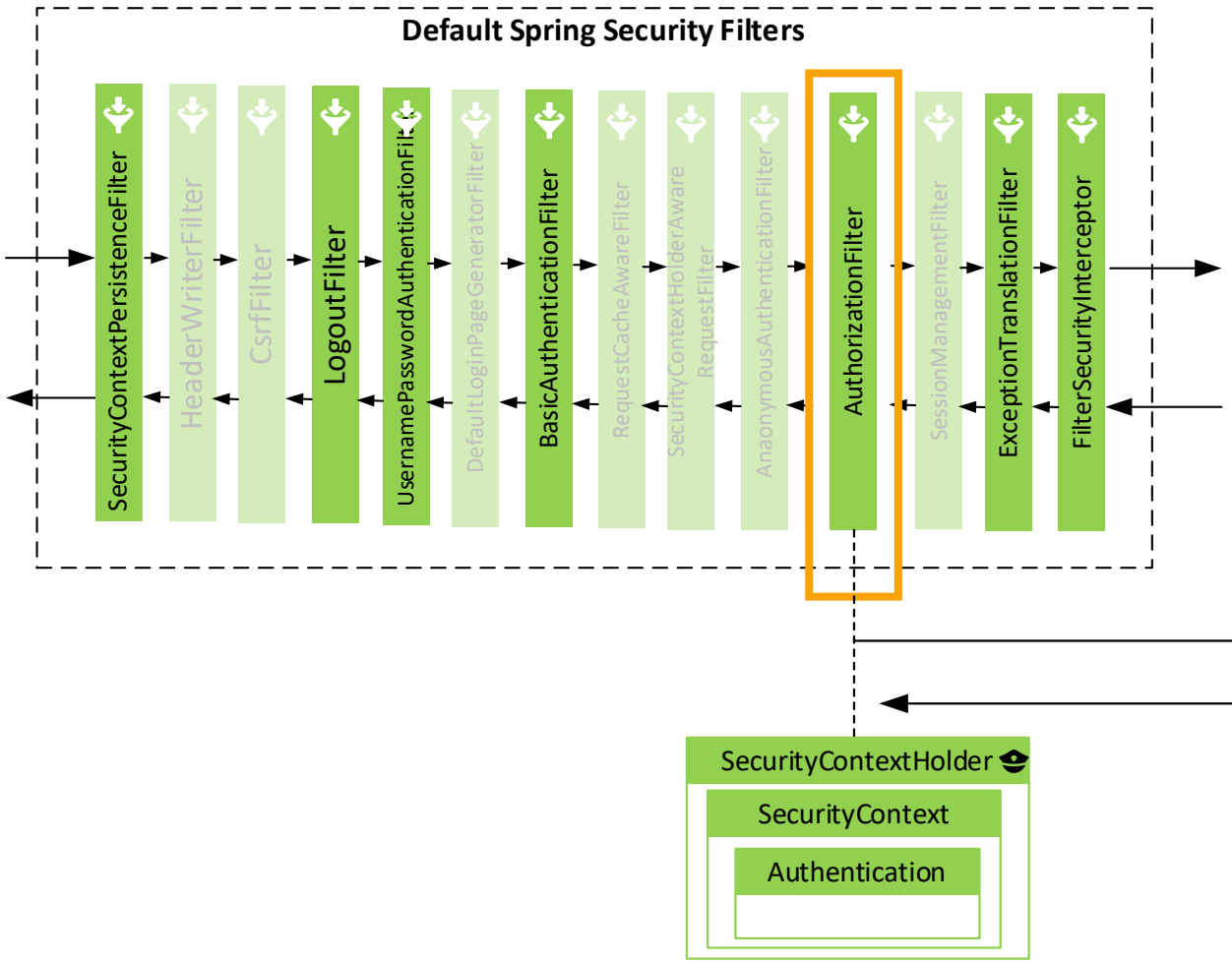
- Mise en place de Token JWT
  - Step5

https://gitlab.com/js-as1/asi1-springboot-and-security

Spring Security Authorization

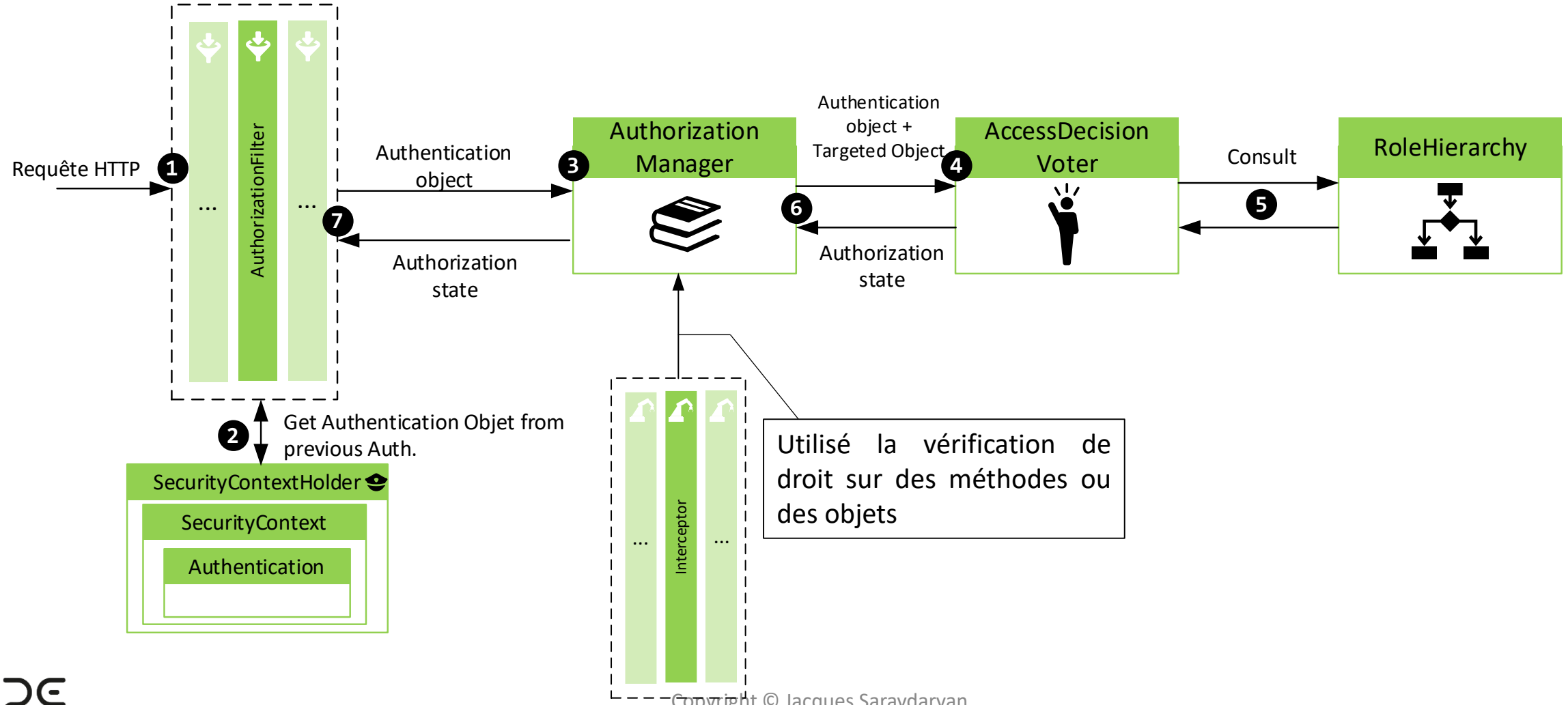


Authorization



- **`AuthorizationFilter:`** restreint les accès aux URLs via un `AuthorizationManager`
- **`AuthorizationManager:`** décide si l'URL est accessible ou non considérant les `Autorities`
- **`AccessDecisionVoter:`** mise en place d'une stratégie de vote sur une l'autorisation d'accès
- **`RoleHierarchy`**: définit un ensemble de règles hiérarchiques sur les `autorities`

Authorization Flow



Role et Authorities

- Creation d'une classe `Role`
- Chaque RoleName correspond à une `Authority`

```java
@Entity
@Table(name = "role")
public class Role {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    public Role(){

         }

    public Role(String s){
        this.roleName=s;
    }

    @Column(name = "role_name")
    private String roleName;

// Getter and Setter
```

User

- Mise à jour de la classe `User`
- Ajout d'une relation entre les tables `@OneToMany`
- Mise à jour de la fonction héritée `getAuthorities()`

```
@Entity
@Table(name = "APPUSER")
public class User implements Serializable,UserDetails{
    private static final long serialVersionUID = 4668602180892212165L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer userId;
    private String username;
    private String password;

    @OneToMany
    (fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private List<Role> roleList;

    public User(){
    }

    @Override
    public List<? extends GrantedAuthority> getAuthorities() {
        List<SimpleGrantedAuthority> roleListAuthorities= new
                                                        ArrayList<>();
        for(Role r: roleList){
            roleListAuthorities.add(new SimpleGrantedAuthority
                                                (r.getRoleName()));
        }
        return roleListAuthorities;
    }

// Getter And Setter
…
```

Role et Authorities

- Mettre à jour `MyUService` pour la création d'un utilisateur avec des rôles

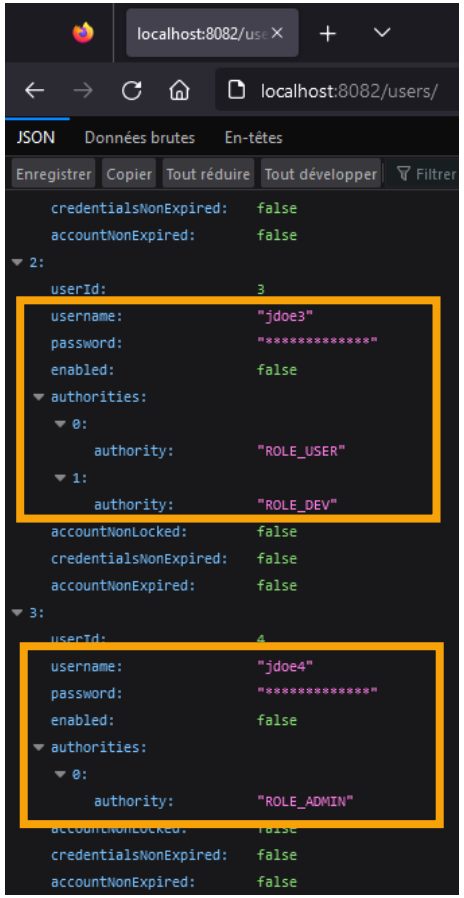
```
@Service
public class MyUService implements UserDetailsService{
    private final UserRepository uRepo;
    private final PasswordEncoder passwordEncoder;
    public MyUService(UserRepository uRepo,PasswordEncoder passwordEncoder)
{
        this.uRepo=uRepo;
        this.passwordEncoder=passwordEncoder;
    }

    public boolean addUser(UserDto userDto) {
        Optional<User> u =uRepo.findUserByUsername(userDto.getUsername());
        if( !u.isPresent()){
            User u_new=new User();
            u_new.setUsername(userDto.getUsername());
            u_new.setPassword(passwordEncoder.encode(userDto.getPassword())
);
            List<Role> authorities=new ArrayList<>();

            userDto.getRoleList().forEach
                                        (e -> authorities.add(new Role(e)));
            u_new.setRoleList(authorities);
            uRepo.save(u_new);
            return true;
        }
        return false;
    }


// …
```

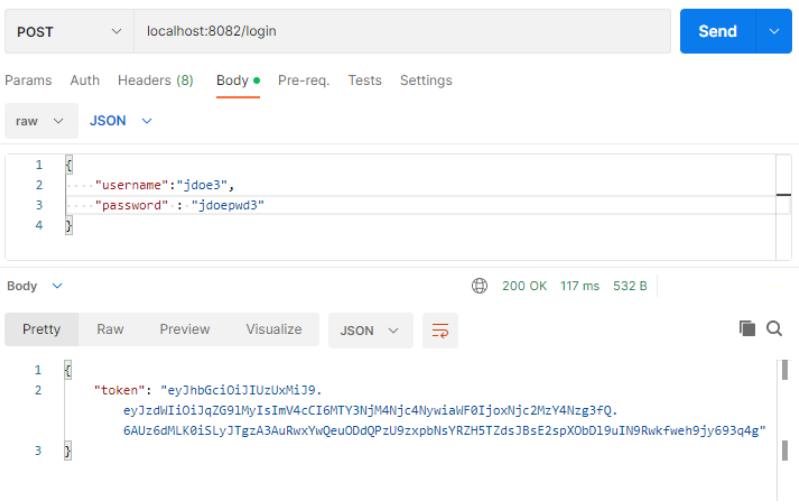
Role et Authorities

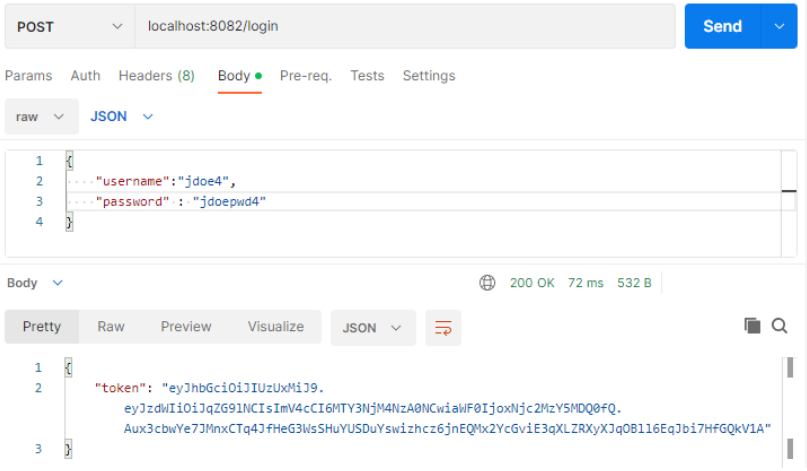
- Modification des règles de filtrage (exemple sur une base d'auth JWT)
- Ajout d'un critère de filtrage `.hasAuthority()` sur une route

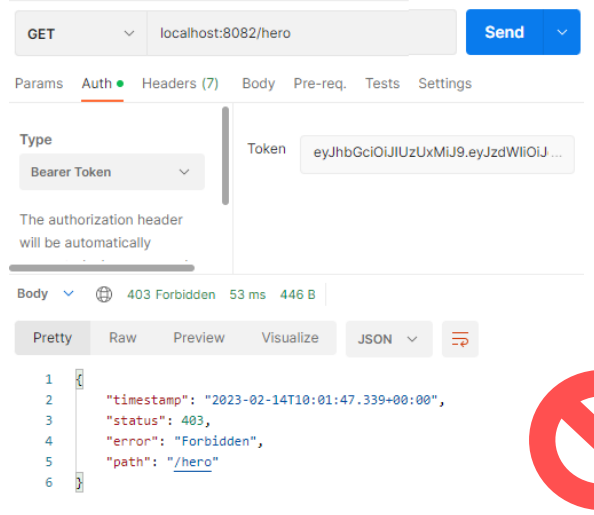
```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    private final MyUService uService;
    private final PasswordEncoder pEncoder;
    @Autowired
    private JwtRequestFilter jwtRequestFilter;
    @Autowired
    private JwtAuthEntryPoint jwtAuthenticationEntryPoint;

    public SecurityConfig(MyUService uService,PasswordEncoder pEncoder) {
        this.uService=uService;
        this.pEncoder=pEncoder;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
        http.csrf(csrf->csrf.disable());
        http.exceptionHandling(ex->ex
            .authenticationEntryPoint(jwtAuthenticationEntryPoint));
        http.sessionManagement(sess->sess
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        http.authenticationProvider(getProvider())
             .authorizeHttpRequests(
                  auth-> auth
                  .requestMatchers("/login").permitAll()
                  .requestMatchers("/hero/**").hasAnyAuthority("ROLE_ADMIN")
                  .anyRequest().authenticated());
        // Add a filter to validate the tokens with every request
        http.addFilterBefore(jwtRequestFilter,
                             UsernamePasswordAuthenticationFilter.class);
        return http.build();
// …
```

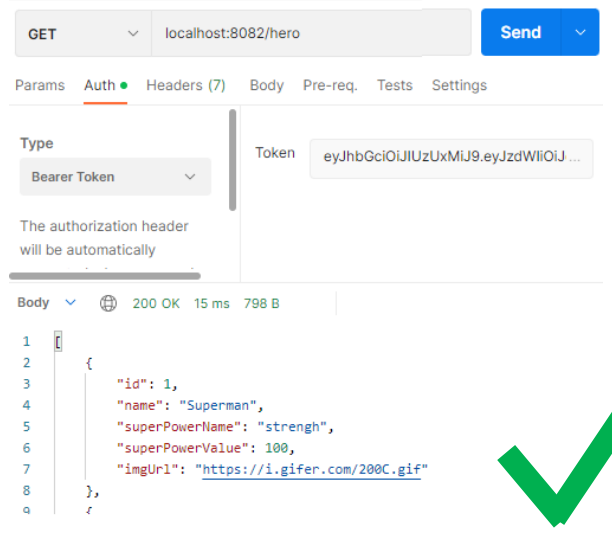
Authorization











A vous de Jouer !

- Mise en place de règles d'authorisation
  - Step6

https://gitlab.com/js-as1/asi1-springboot-and-security

✉ **Jacques Saraydaryan**
Jacques.saraydaryan@cpe.fr