SUPSI

Session, Cookies, Authentication e Spring Security

Protocollo HTTP è stateless

- Il protocolo HTTP è un protocollo detto stateless, ossia **senza stato**. Ogni richiesta è indipendente dall'altra e non ha nessuna relazione con richieste precedenti
- In molti casi però ci servirebbe tenere uno stato dell'applicazione, per esempio:
  - Per personalizzare l'informazione (contenuto, localizzazione, …)
  - Mantenere alcuni dati tra richieste durante la navigazione, possibilmente anche tra visite diverse (siti di e-commerce, shopping cart)
  - Collezionare informazioni sull'uso dell'applicazione/sito
  - …
- L'esempio più evidente è durante l'autenticazione ad un sito: una volta autenticato, le prossime richieste HTTP dell'utente non dovrebbero causare la richiesta da parte del server del nome utente e della password un'altra volta

Cookie

- I cookie sono stringhe di testo di piccola dimensione (4KB) inviate da un server ad un Web client e poi rimandati indietro dal client al server ogni volta che il client accede allo stesso dominio web
- Un cookie è un header aggiuntivo presente in una richiesta (*Cookie:*) o risposta (*Set-cookie:*) .
- Esempi:
  - SET-COOKIE: name=value
  - SET-COOKIE: name=value; Expires=Wed, 09 Jun 2021 10:18:14 GMT
  - Da parte del browser il cookie viene passato da lì in avanti con tutte le richieste: COOKIE: name=value; name2=value2
- Aggiunge uno stato al protocolo HTTP che di fatto stato non ce l'ha

Cookie

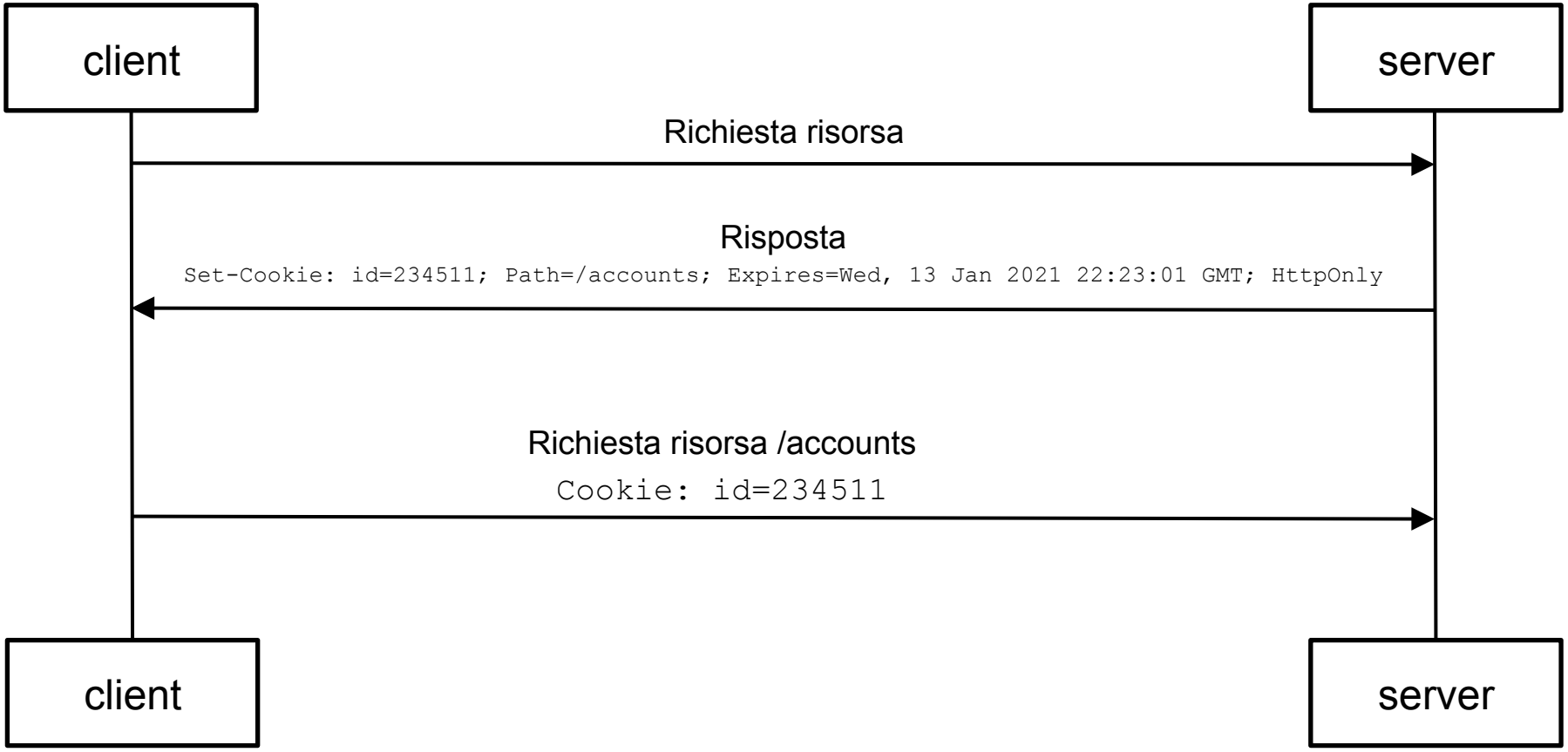
- Nel cookie solitamente possiamo trovare i seguenti attributi:
  - **Nome/valore**: è un campo obbligatorio
  - **Scadenza** è un attributo opzionale che permette di stabilire la data di scadenza del cookie
  - **Modalità d'accesso** (HttpOnly) rende il cookie invisibile a javascript e altri linguaggi client-side presenti nella pagina
  - **Sicuro** (secure) indica se il cookie debba essere trasmesso criptato con HTTPS
  - **Dominio e percorso** definiscono l'ambito di visibilità del cookie, indicano al browser che il cookie può essere inviato al server solo per il dominio e il percorso indicati

Cookie - Esempio response header HTTP

- Cache-Control no-cache
- Content-Encoding gzip
- Content-Type application/json
- DateTue, 03 Nov 2015 09:34:35 GMT
- Expires0
- Set-Cookie

  session-id=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT session-id-time=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT session-token=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT ubid-acbit=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT at-acbit=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT lc-acbit=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT x-acbit=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT x-wl-uid=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT sess-at-acbit=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT UserPref=-; path=/; domain=.www.amazon.it; expires=Mon, 03-Nov-2003 09:34:35 GMT ubid-acbit=280-9154472-9055434; path=/; domain=.amazon.it; expires=Tue, 01-Jan-2036 00:00:01 GMT lc-acbit=it_IT; path=/; domain=.amazon.it; expires=Tue, 01-Jan-2036 00:00:01 GMT session-id-time=2082758401l; path=/; domain=.amazon.it; expires=Tue, 01-Jan-2036 00:00:01 GMT session-id=277-7995917-6750516; path=/; domain=.amazon.it; expires=Tue, 01-Jan-2036 00:00:01 GMT

Cookie


client
server
Richiesta risorsa
Risposta
Set-Cookie: id=234511; Path=/accounts; Expires=Wed, 13 Jan 2021 22:23:01 GMT; HttpOnly
Richiesta risorsa /accounts
Cookie: id=234511
client
server

Alcuni esempi di cookie

- www.cnn.ch

  Set-Cookie:CG=CH:20:Manno; path=/

- www.tinext.com

  Set-Cookie:BALANCER=Balancer.magnoprod01; path=/;

- www.amazon.it

  Set-Cookie:session-id=278-7644526-5656418; path=/; domain=.amazon.it; expires=Tue, 01-Jan-2036 00:00:01 GMT

  Set-Cookie:session-id-time=-; path=/; domain=.www.amazon.it; expires=Mon, 30-Sep-2002 14:46:50 GMT

  Set-Cookie:session-id-time=2082758401l; path=/; domain=.amazon.it; expires=Tue, 01-Jan-2036 00:00:01 GMT

Tipi di Cookie

- **Session Cookie** (in-memory cookie or transient)
  - Esiste solo in memoria temporanea mentre l'utente naviga un website e sono cancellati dal terminale dell'utente quando il browser è chiuso.
- **Persistent Cookie** (tracking cookie)
  - Dura di più della sessione dell'utente, non è cancellato una volta che il browser è chiuso
- **Secure cookie**
  - Solo usato via https, cookie sempre criptato quando trasmesso da client a server (limita attacchi di man-in-the-middle)
- **Httponly Cookie**
  - Supportato dai browser più moderni. Sarà usato solo quando si trasmettono richieste http(s), quindi limitando l'accesso al cookie da NON-HTTP API, come da javascript (limitando quindi la possibilità di attacchi cross-site scripting (XSS))

Alternative al cookie

- **Tracking dell'indirizzo ip**

  Per esempio nei sondaggi online, molti sondaggi sanno che hai già votato. Come? Registrano l'indirizzo ip del client al momento del voto
  - Problema: molti computer dall'esterno hanno lo stesso indirizzo ip

- **URL (query string)** il server genera dei link all'interno delle pagine dove nella query string vengono concatenate le informazioni

- **Hidden form fields** il server nasconde le informazioni in campi hidden (non visibili quindi)

- **storage appositi sul browser (ma non tutti i client li hanno):**
  - **sessionStorage, localStorage**
  - **indexedDb (nosql db)**: usabilità simile a un database
  - **webSql** non è più mantenuto nelle specifiche del w3c

Third-party cookie

- I "Third party cookies" sono creati e utilizzati da un ente diverso dal proprietario del sito che si sta visitando.

- Per esempio, un sito Internet per analizzare la propria utenza può ricorrere ai servizi di una società esterna.

- Contenuti integrati come ads (pubblicità mirata), youtube, slide show di Flickr, per esempio possono settare dei propri cookie

Esempio Cookie nelle servlet

```
Cookie cookie = new Cookie("myCookie", "myCookieValue");
cookie.setMaxAge(24 * 60 * 60);  // 24 hours
response.addCookie(cookie);

Cookie[] cookies = request.getCookies();

String userId = null;
for(Cookie cookie : cookies){
  if("uid".equals(cookie.getName())){
      userId = cookie.getValue();
  }
}
```

Session

- Tenere traccia degli utenti durante la navigazione su un determinato sito viene definita una sessione web
- Una sessione web solitamente inizia caricando un applicazione web sul browser e termina chiudendo il browser. Ma può essere anche più lunga.
- Esempi:
  - tenere traccia di un utente su un sito di e-commerce
  - mantenere autenticato un utente
  - …

Session

- Gli application server hanno una loro gestione delle sessioni, viene sempre creato un SessionId per ogni sessione:
  - SessionId viene messo in un cookie, oppure
  - SessionId viene passato nel URL come query string, o hidden fields

- Le informazioni legate ad un sessione sono:
  - salvate in memoria, ma limitata nel tempo e se il server cade, cade la sessione (Tomcat crea un file .ser per salvarla), oppure
  - salvate su un DB o su file

- Il SessionId dovrebbe essere criptato per non essere facilmente indovinato
- Le sessioni scadono dopo un periodo di inattività

Esempio HttpSession nelle servlet

```
HttpSession session = request.getSession();

session.setAttribute("user", new User("michi", "Michele",
"Bernasconi"));
```

Nella risposta viene creato il cookie JSESSIONID:

```
Cookie: JSESSIONID=8BE9D3153A0FE2A4E0D8A88E9D7E976D
```

Il client conosce solo l'id della sessione, mentre lato server, associato a quel id, vi può essere qualsiasi oggetto. Un oggetto HttpSession è una sorta di dizionario, è un contenitore di coppie chiave-valore.

Authentication vs Authorization

- L'**autenticazione** è il processo di generazione, trasmissione e verifica di credenziali (username e password nel caso di un utente) per stabilire l'identità delle parti.

- L'**autorizzazione** indica il permesso di accedere alle risorse da parte degli utenti.

  Per esempio: L'utente appena autenticato ha il permesso di accedere alla pagina di amministrazione del sistema?

Authentication

- Permette l'accesso a una pagina web solo se l'utente ha fornito il corretto nome utente e password
- Ci sono almeno tre tipi di autenticazione:
  - Basic access authentication (BASIC)
  - Digest access authentication (DIGEST)
  - Form based authentication (FORM)
    - Captcha, reCaptcha

BASIC Authentication

- Il server richiede:

```
401 Authorization Required
www-Authenticate: Basic realm="authenticate yourself"
```

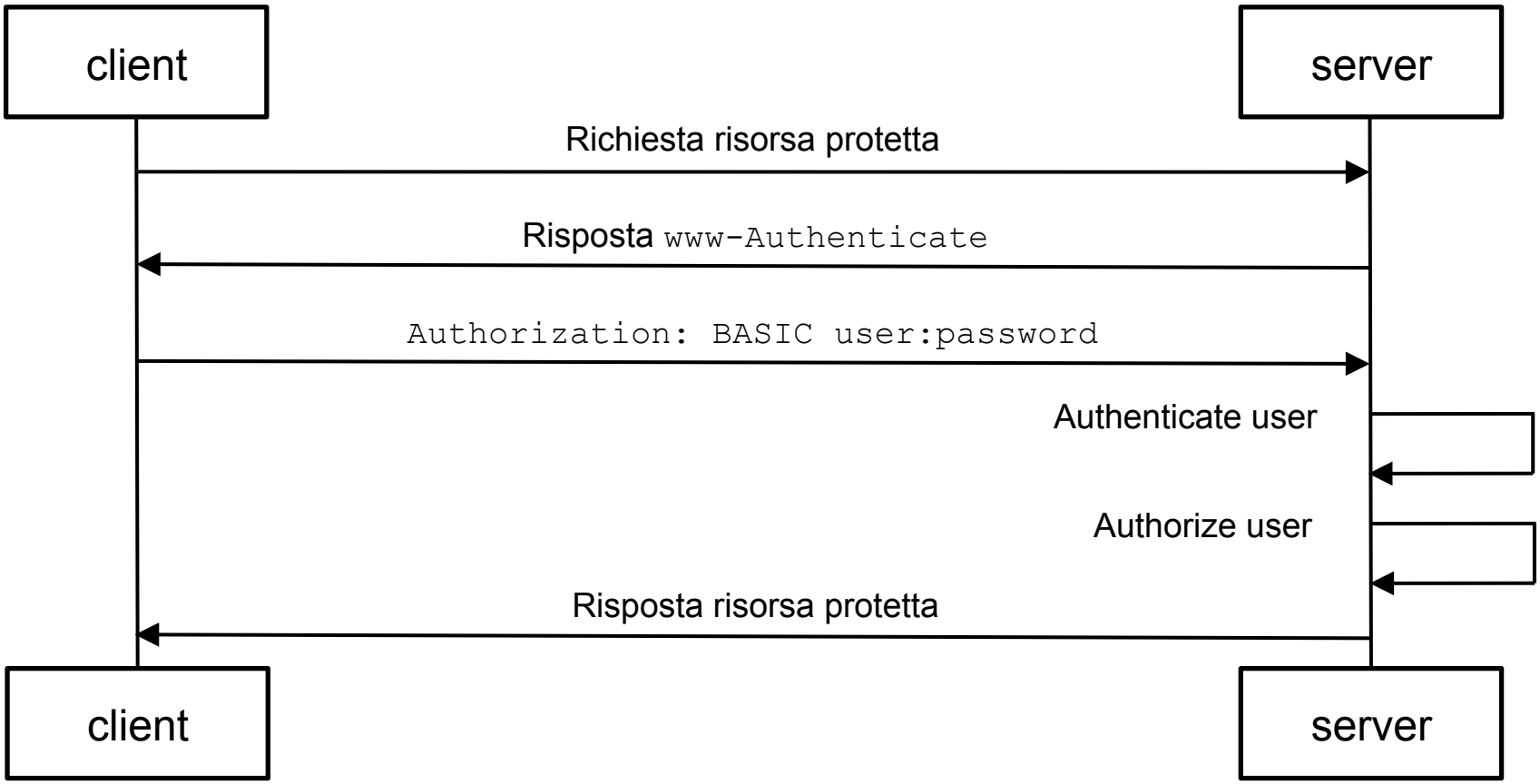
- Prima di trasmettere lo username e la password immessi dall'utente, le due stringhe sono concatenate con un ":" che separa i due valori. La stringa viene poi codificata Base64. Non per criptarla, ma piuttosto per eliminare caratteri non compatibili con HTTP

- Il client risponde:

```
Authorization: Basic bWFyY286bWlhcGFzc3dvcmQ=
```

- Le successive richieste conterranno l'header `Authorization`

BASIC Authentication



DIGEST Authetication

- Il server richiede:

```
401 unauthorized
WWW-Authenticate: Digest realm="autenticati",
qop="auth",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

- Applica una funzione hash a **user-password-realm-nonce-URI** prima di inviarla sulla rete. Questo meccanismo è più sicuro dell'invio mediante BASIC authentication, che invia la password in testo semplice.

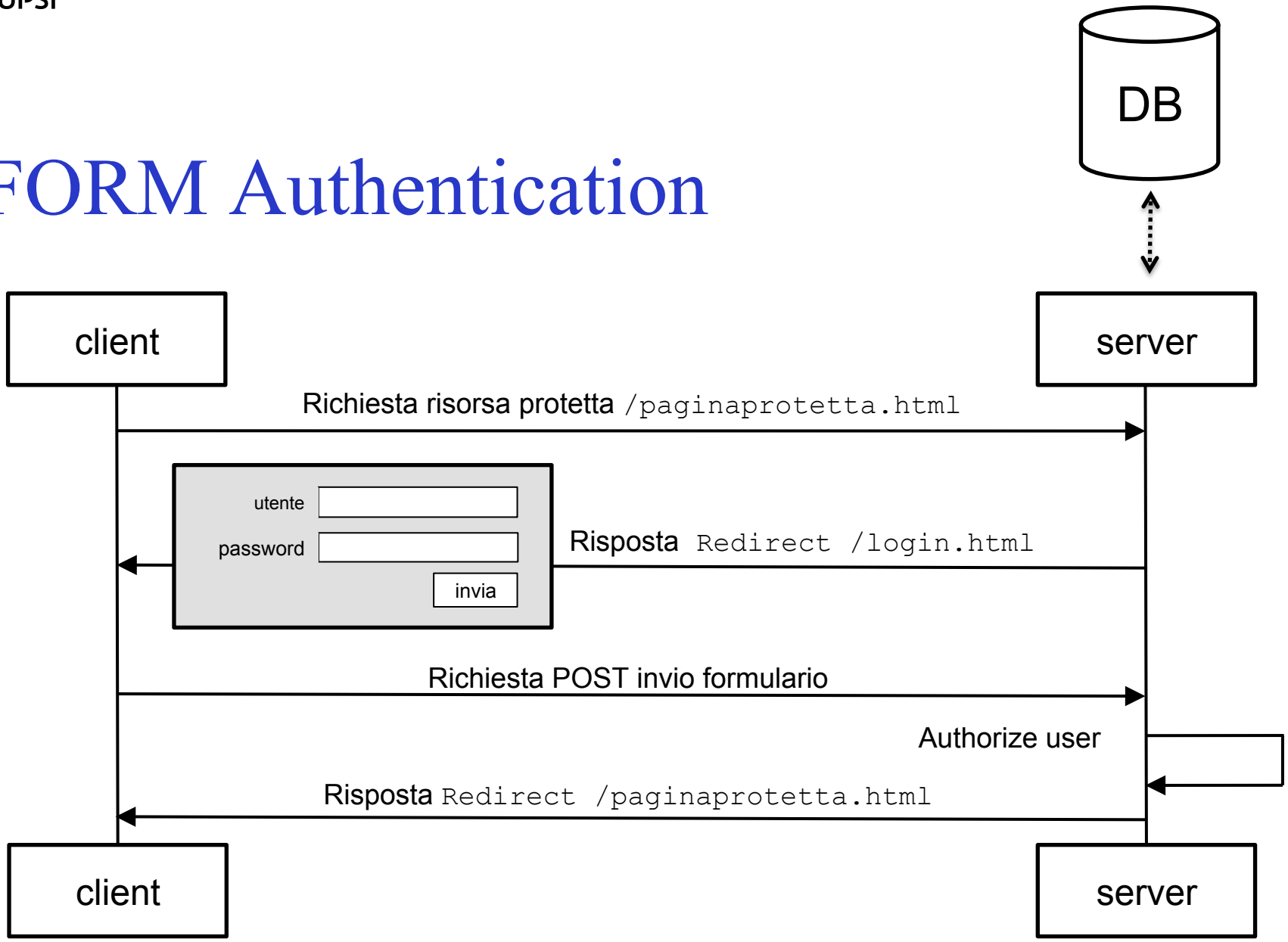
- `nonce` è un “one-time number” generato dal server, questo permette di eludere dei “replay-attack”.

FORM Authentication

- Il modo più utilizzato, ma richiede un po' di lavoro supplementare per essere implementato
- All'utente viene presentato un form all'interno di una pagina web. L'autenticazione viene integrata meglio nell'applicazione.
- Questo metodo soffre, però, di alcuni rischi di sicurezza come per la BASIC authorization: le credenziali di accesso sono trasmesse *in chiaro*. Usare https per ovviare a questo problema.
- Codice html per mostrare un form di autenticazione:

```
<form method="post" action="/secured/login">
    <input type="text" name="username">
    <input type="password" name="password">
    <input type="submit" value="Invia">
</form>
```

FORM Authentication



Spring Security

- Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements

- Comprehensive and extensible support for both Authentication and Authorization
- Protection against attacks like session fixation, clickjacking, cross site request forgery, ...
- Servlet API integration
- Optional integration with Spring Web MVC
- Much more…

- Da https://projects.spring.io/spring-security/

Integrazione Spring Boot

- pom.xml

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

- L'autenticazione è gestita tramite un oggetto **utente** che ha principalmente un **username** e una **password** (criptata).
- L'autorizzazione invece viene gestita con il **ruolo associato all'utente**.
- I ruoli, per convenzione, in Spring Security iniziano con "ROLE_", per esempio ROLE_ADMIN

Customizzare Spring Security

- Implementare `UserDetailsService`

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

@Service
public class CustomUserDetailService implements UserDetailsService {

   @Autowired
   private BlogService blogService;

   @Override
   public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
      Utente user = blogService.findUserByUsername(username);
      if (user == null) {
         throw new UsernameNotFoundException("User not found");
      }
      List<GrantedAuthority> auth = AuthorityUtils.createAuthorityList(user.getRole().getName());
      return new User(username, user.getPassword(), true, true, true, true, auth);
   }

}
```

- Da notare che l'utente ritornato dal metodo loadUserByUsername è di tipo **`org.springframework.security.core.userdetails.User`**

Configurare Spring Security

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
         .authorizeRequests()
            .antMatchers("/", "login").permitAll()
            .antMatchers("/blog/new").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and()
         .formLogin()
            .loginPage("/login")
            .failureUrl("/login?error")
            .and()
         .logout()
            .logoutUrl("/logout")
            .logoutSuccessUrl("/");
      http.csrf().disable();
   }

   (…) // continua prossima slide

}
```

Attenzione! Sul DB la stringa del ruolo è ROLE_ADMIN

- Vengono configurate quali risorse sono accessibili con o senza autenticazione

Configurare Spring Security

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

   @Autowired
   private CustomUserDetailService customUserDetailService;

   (…)

   @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
      DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
      authProvider.setUserDetailsService(customUserDetailService);
      authProvider.setPasswordEncoder(new BCryptPasswordEncoder());

      auth.authenticationProvider(authProvider);
    }

}
```

- Configura l'autenticazione da DB. Viene anche configurata la funzione di hashing da usare per criptare le password (BCrypt).

Form login

- Spring Security aggiunge l'endpoint **/login in POST** automaticamente. E si aspetta di poter leggere i parametri **username** e **password** dalla richiesta.

- La gestione dell'endpoint **/login in GET** deve essere gestita dallo sviluppatore

```
<form method="post">

 <input type="text" id="title" id="username" name="username">
 <input type="password" id="password" name="password" >

 <input type="submit" value="Invia">

</form>
```

Accedere a utente loggato da java

```
User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
blogPost.setAuthor(blogService.findUserByUsername(user.getUsername()));
```

- User è di tipo **`org.springframework.security.core.userdetails.User`**

- Oppure tramite la sessione

```
@PostMapping("/blog/new")
public String post(BlogPost blogPost, HttpSession session){
     SecurityContextImpl sc = (SecurityContextImpl)session.getAttribute("SPRING_SECURITY_CONTEXT");
     User user = (User) sc.getAuthentication().getPrincipal();
     blogPost.setAuthor(blogService.findUserByUsername(user.getUsername()));
     …
}
```

Thymeleaf Spring Security integration

- pom.xml

```
<dependency>
	<groupId>org.thymeleaf.extras</groupId>
	<artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

- Aggiunge un nuovo namespace:

```
<html xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
```

Esempio usando expression utility objects

- Authentication:

```
<p>Sei loggato come <b th:text="${#authentication.name}">user</b></p>
```

- Authorization:

```
<div th:if="${#authorization.expression('hasRole(''ROLE_ADMIN'')')}">
    Sarà visualizzato solo se l'utente autenticato ha il ruolo ROLE_ADMIN
</div>
```

Esempio usando attributi sec:

- Authentication:

```
<div sec:authentication="name">
  il nome dell'utente autenticato apparirà qui
</div>
```

- Authorization:

```
<div sec:authorize="hasRole('ADMIN')">
   Sarà visualizzato solo se l'utente autenticato ha il ruolo ROLE_ADMIN
</div>

Solo se l'utente è autenticato (ha fatto il login)
<a sec:authorize="isAuthenticated()" href="/logout">logout</a>

Solo se l'utente può raggiungere l'url /blog/new
<a sec:authorize-url="/blog/new" th:href="@{/blog/new}">Nuovo post</a>
```

Links

- https://projects.spring.io/spring-security/
- https://spring.io/guides/gs/securing-web/