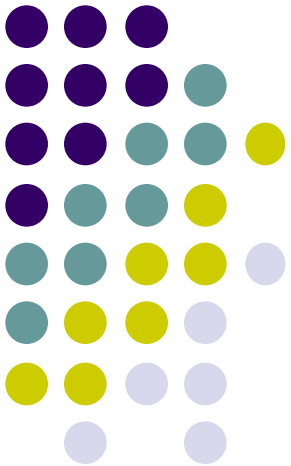
Мова програмування Java та технології J2EE

Безпека

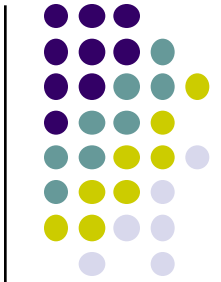


Сирота О.П.

Тема лекції

- Основні поняття безпеки
- Java EE Security
  - Приклад сценарію аутентифікації та авторизації для веб-застосування
  - Аутентифікація
  - Авторизація
  - Декларативна безпека
  - Програмна безпека
- Spring Security
  - AuthenticationManager
  - Security Web Resources
  - Securing methods

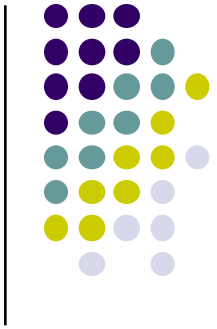
Основні поняття безпеки



Ваші варіанти?

- **Аутентифікація**
- **Авторизація**
- **Конфіденційність**
- **Цілісність**
- **Аудит**

Основні поняття безпеки



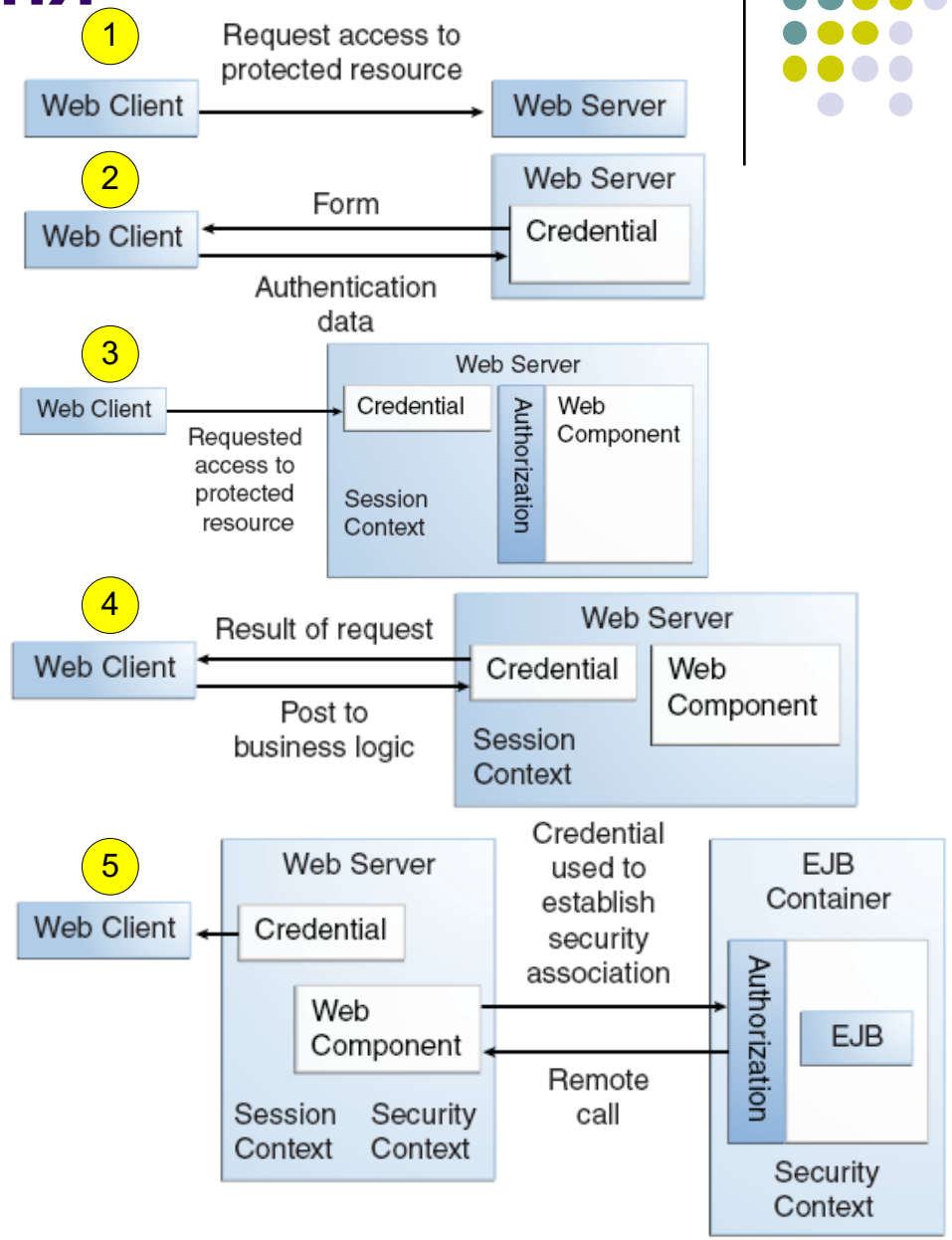
- **Аутентифікація** – перевірка користувача (що він дійсно є тим, за кого себе видає)
- **Авторизація** – перевірка, що користувач має доступ до захищеного ресурсу (чи йому дозволено виконувати запрошену функцію)
- **Конфіденційність** – захист даних від перегляду третіми особами під час передачі даних
- **Цілісність** – забезпечення, що отримані дані є тими даними, які були відправлені
- **Аудит** – можливість відстежувати усі рішення з безпеки (аутентифікація, авторизація) з метою у подальшому переглянути та проаналізувати настройки безпеки

Які компоненти потребують захисту доступу в Java EE?

- Необхідно захищати наступні компоненти:
  - Веб-компоненти: ресурс за URL, REST
  - Методи EJB компонентів (EJB out of scope in this lecture)

Приклад сценарію аутентифікації для веб-застосування

- Крок 1
  - запит захищеного ресурсу (захищений ресурс визначається по URL)
- Крок 2
  - аутентифікація (перевірка логіну та паролю)
- Крок 3
  - авторизація (перевірка доступу до запрошеного ресурсу)
- Крок 4
  - виконання початкового запиту
- Крок 5
  - виклик методу EJB в процесі виконання бізнес-логіки



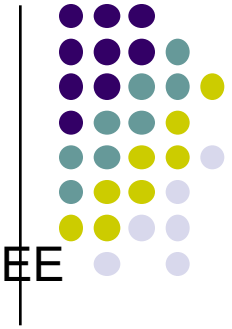
Безпека в Java SE

- Механізми безпеки в Java SE:
  - Java Authentication and Authorization Service (JAAS)
  - JavaGeneric Security Services (Java GSS-API)
  - Java Cryptography Extension (JCE)
  - Java Secure Sockets Extension (JSSE)
  - Java Simple Authentication and Security Layer API (Java SASL API)

Безпека в Java EE

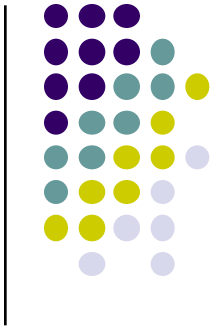
- Механізми безпеки в Java EE поділяються на наступні 3 види:
  - Безпека на рівні застосування
    - Доступ до компонентів веб-модулів, EJB-модулів
  - Безпека на рівні транспорту
    - HTTPS
  - Безпека на рівні повідомлень (для веб-сервісів)
    - Безпека для SOAP-повідомлень. Не є частиною стандарту Java EE

Безпека в Java EE

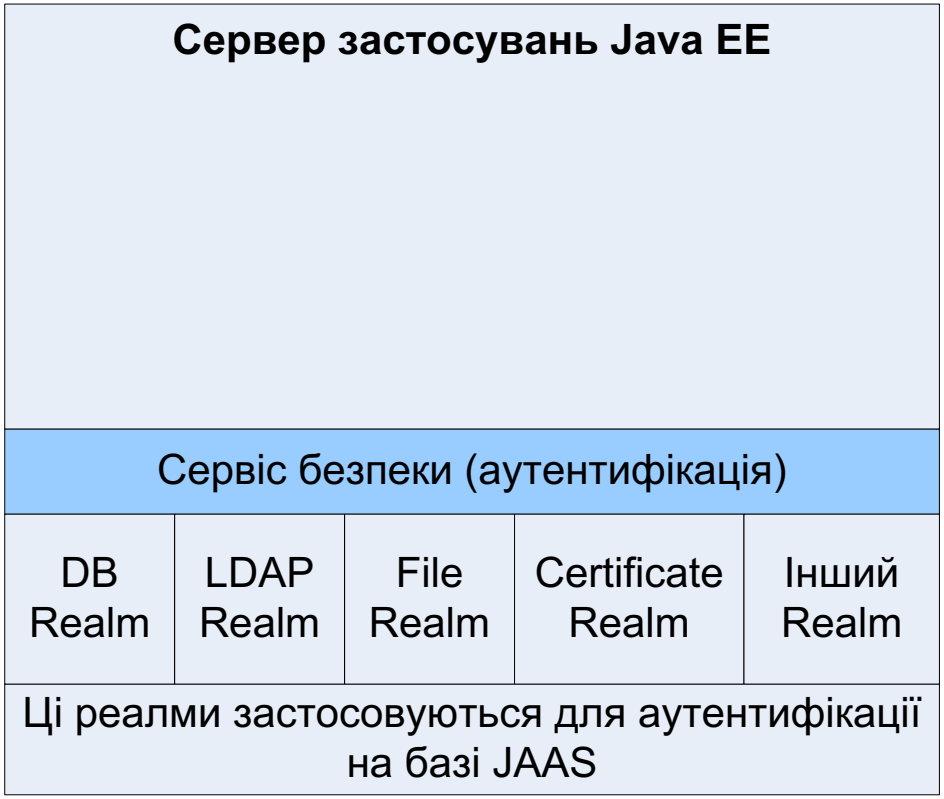


- Механізми безпеки, які реалізуються сервером застосувань Java EE
  - Безпека на рівні застосування:
    - Аутентифікація – забезпечується сервером застосувань (сервісом безпеки)
    - Авторизація – забезпечується контейнерами (веб-контейнером, EJB-контейнером)
  - Транспорт
    - Secure HTTP – забезпечується HTTP-сервісом
    - Secure IIOP (для виклику EJB-компонентів, IIOP over SSL) – забезпечується ORB-сервісом
  - Безпека на рівні повідомлень (для веб-сервісів)
    - Цей рівень безпеки підтримується кожним сервером застосувань, але не є частиною стандарту Java EE. Glassfish використовує для безпеки веб-сервісів стандарти WSS (Web Service Security)
- Інфраструктура – настройка сервісу безпеки
  - Підключення нових реалмів безпеки
  - Підключення модулів, які пов'язують безпеку на рівні застосування із аутентифікаційною інформацією. Базується на Java Authorization Contract for Containers (JACC)
  - Аудит рішень з безпеки, які прийняті сервісом безпеки

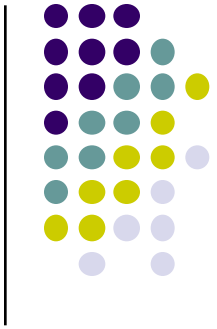
Аутентифікація у Java EE сервері застосувань



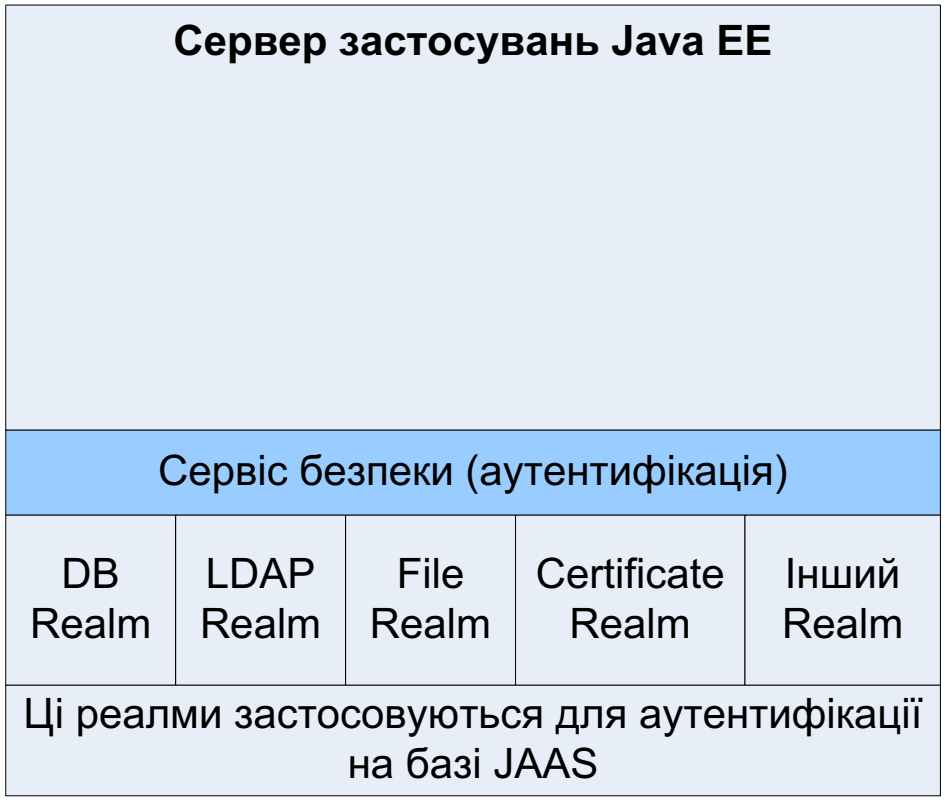
- Аутентифікацію забезпечує сервер застосувань
- Алгоритм аутентифікації:
  - Пошук користувача по логіну
  - Перевірка пароля
  - Пошук груп, до яких відноситься користувач
- Аутентифікація забезпечується сервісом безпеки - на основі реалмів
- Реалм – це область настройок безпеки, яка визначена для сервера застосувань



Реалми безпеки



- Можуть бути визначені наступні реалми для аутентифікації:
  - БД
  - Directory Server (LDAP)
  - Файл
  - Сховище сертифікатів
  - Можуть бути підключені інші реалми (наприклад, які делегують виконання аутентифікації зовнішній системі аутентифікації)



Зовнішня аутентифікація

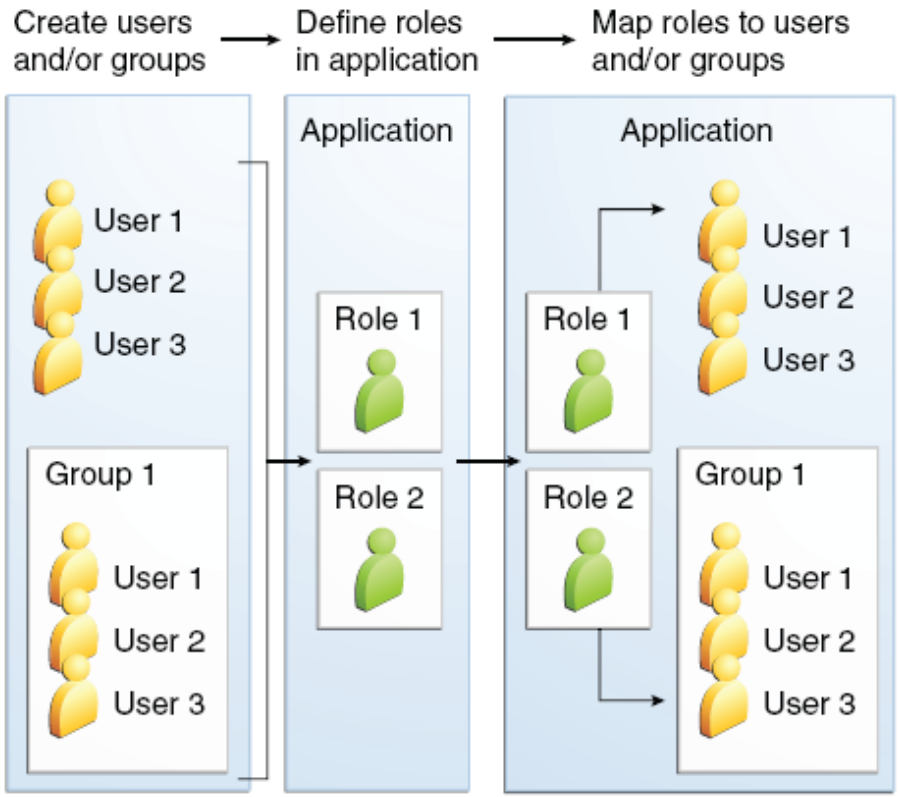
- При застосуванні стандартних реалмів аутентифікації застосовуються потужності сервера застосувань
  - Це може бути недопустимо для високонавантажених систем, які витримують тисячі логінів в секунду – оскільки потужності частково зайняті задачами аутентифікації, а не основною бізнес-логікою
- Переваги застосування зовнішньої системи аутентифікації:
  - Зниження навантаження на прикладну систему (зовнішня система працює на власних технічних потужностях)
  - Інтеграція з SSO (Single Sign On)
- Для інтеграції із зовнішньою системою аутентифікації використовується протокол SASL (Simple Authentication and Security Layer)

Авторизація у Java EE сервері застосувань

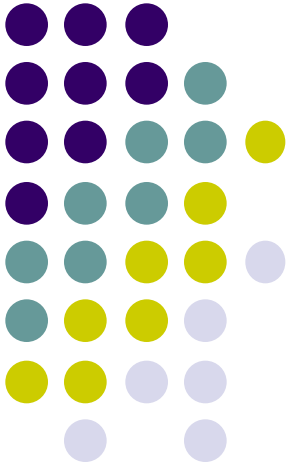
- Авторизація забезпечується контейнером
- Алгоритм авторизації:
  - Перевірка доступу до захищених ресурсів на основі **ролей**
- **Ролі**
  - Роль – це абстракція, яка задається на рівні застосування, та визначає набір прав доступу
  - Структура ролей в Java EE – плоска (без ієрархій)
  - Користувач може належати до декількох ролей
- Відповідність між ролями та групами - задається на рівні застосування

Користувачі, групи, ролі

- Ви створюєте **користувачів** та об'єднуєте їх у **групи**
  - у БД, файлах, Directory Server…
- На рівні Вашого застосування:
  - визначаєте **ролі**
  - задаєте **відповідність ролей та груп**
- При аутентифікації:
  - Сервер застосувань перевіряє користувача
  - Сервер застосувань визначає групи, до яких належить користувач
- При авторизації
  - Контейнер визначає ролі користувача на основі інформації про групи та заданої Вами інформації щодо відповідності ролей та груп
  - Контейнер (чи ваш код) перевіряє доступ ролі до ресурсу



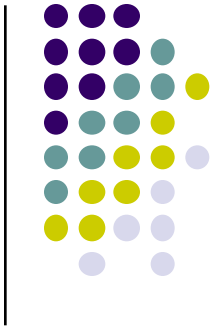
Безпека веб-застосувань



Попередня настройка

- Обов’язково необхідно:
  - Сервер застосувань: настроїти відповідний **реалм безпеки** для аутентифікації
  - Веб-застосування: визначити для веб-застосування **ролі та відповідність між ролями та групами** для авторизації

Приклад настройки JDBC Realm



- Glassfish, http://localhost:4848
- Security -> Realms -> New Realm

- Властивості JDBC Realm
  - JNDI = JNDI-name джерела даних
  - Jaas Context = jdbcRealm
  - **User Table**
  - **User Name Column**
  - **Password Column**
  - **Group Table**
  - **Group Name Column**
  - **Digest Algorithm**

- Див. приклад у таблицях ->

* Якщо Ви не використовуєте Digest для паролів, встановіть в полі значення NONE

USERTABLE

USERID	PASSWORD
Ivanov	****

GROUPTABLE

USERID	GROUPID
Ivanov	studentGroup

JDBC REALM

User Table	USERTABLE
User Name Column	USERID
Password Column	PASSWORD
Group Table	GROUPTABLE
Group Name Column	GROUPID
Digest Algorithm*	MD5

Визначення ролей веб-застосування

- **Ролі** визначаються у веб-застосуванні у дескрипторі розгортання або за допомогою анотацій
  - У дескрипторі розгортання **web.xml**.

```
<security-role>
    <role-name>student</role-name>
</security-role>
<security-role>
    <role-name>teacher</role-name>
</security-role>
```

    - Якщо Ви застосовуєте Netbeans, то він надає графічний інтерфейс для завдання ролей (відкрити web.xml, закладка “Security”, секція Security Roles)

  - За допомогою анотації **@DeclareRoles**

```
@DeclareRoles({"student", "teacher"})
public class TestServlet extends HttpServlet {…}
```

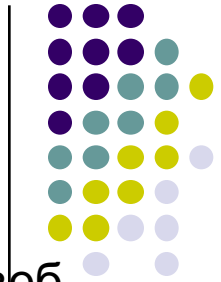
Java EE 6 only

- **Відповідність між ролями та групами** визначається у веб-застосуванні у специфічному для сервера застосувань дескрипторі розгортання (для Glassfis – це **sun-web.xml)**.

```
<security-role-mapping>
    <role-name>student</role-name>
    <group-name>studentGroup</group-name>
</security-role-mapping>
<security-role-mapping>
    <role-name>teacher</role-name>
    <group-name>teacherGroup</group-name>
</security-role-mapping>
```

  - Якщо Ви застосовуєте Netbeans, то він надає графічний інтерфейс для завдання відповідності ролей (відкрити sun-web.xml, закладка “Security”, кнопка Add Security Role Mapping)

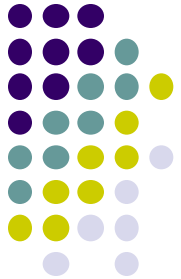
Декларативна vs програмна безпека



Оберіть спосіб виконання аутентифікації та авторизації для Вашого веб-застосування

- **Декларативний**. У цьому разі безпекою управляє контейнер. Програміст задає відповідні настройки у дескрипторі розгортання чи анотаціях.
- **Програмний**. Програміст розробляється програмний код. Застосовується, якщо недостатньо декларативних засобів.

Декларативне управління безпекою веб-застосувань



- Декларативне завдання аутентифікації
  - Сервер виконує аутентифікацію згідно настройок
  - Програміст вказує у настройки для логіну (реалм та метод логіну). Настройки вказуються у дескрипторі розгортання.

- Декларативне завдання авторизації
  - Сервер виконує контроль доступу згідно настройок
  - Програміст вказує ресурси, які треба захищати (у дескрипторі розгортання або у анотаціях)

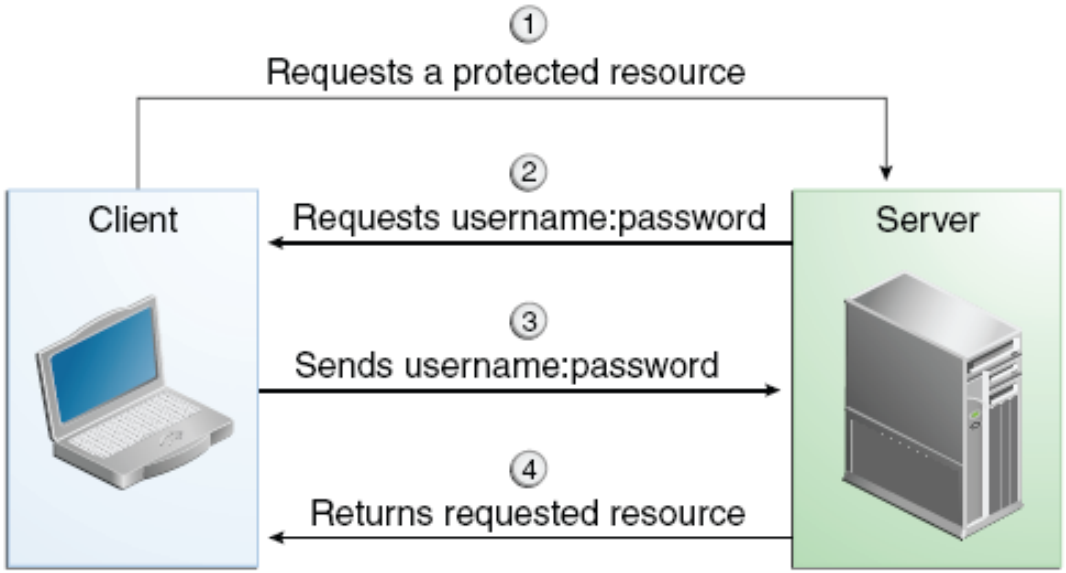
Методи логіну для веб-застосування

- Нижче вказане відноситься до декларативного управління аутентифікацією
- Платформа Java EE підтримує наступні методи логіну:
  - NONE, BASIC, DIGEST, FORM, CLIENT-CERT
- Кожний тип має власні правила конфігурування
- Одне веб-застосування може підтримувати тільки один метод логіну

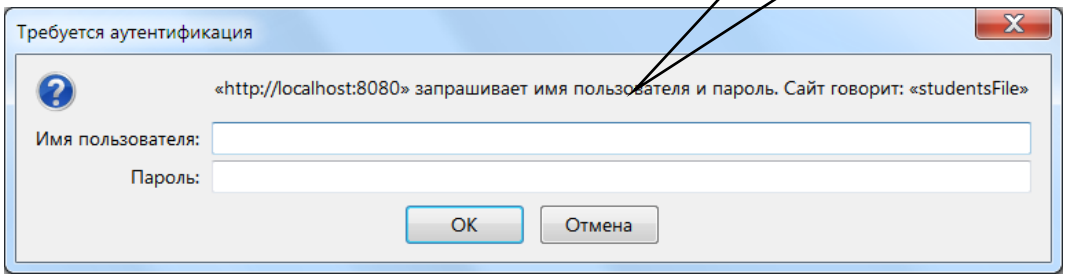
Метод логіну	Конфігурування
BASIC	• Вказати цей тип в web.xml
CLIENT-CERT	• Вказати цей тип в web.xml • База даних сертифікатів сервера застосувань (для Glassfish це certificate realm)
FORM	• Вказати цей тип в web.xml • Розробити сторінки логіна та помилки аутентифікації • Вказати ці назви у web.xml
DIGEST	• Вказати цей тип в web.xml • Підтримується не усіма броузерами та серверами
NONE	

BASIC

- Для вводу логіну та паролю броузер надає вспливаюче вікно
- Пароль передається у зашифрованому вигляді Base64

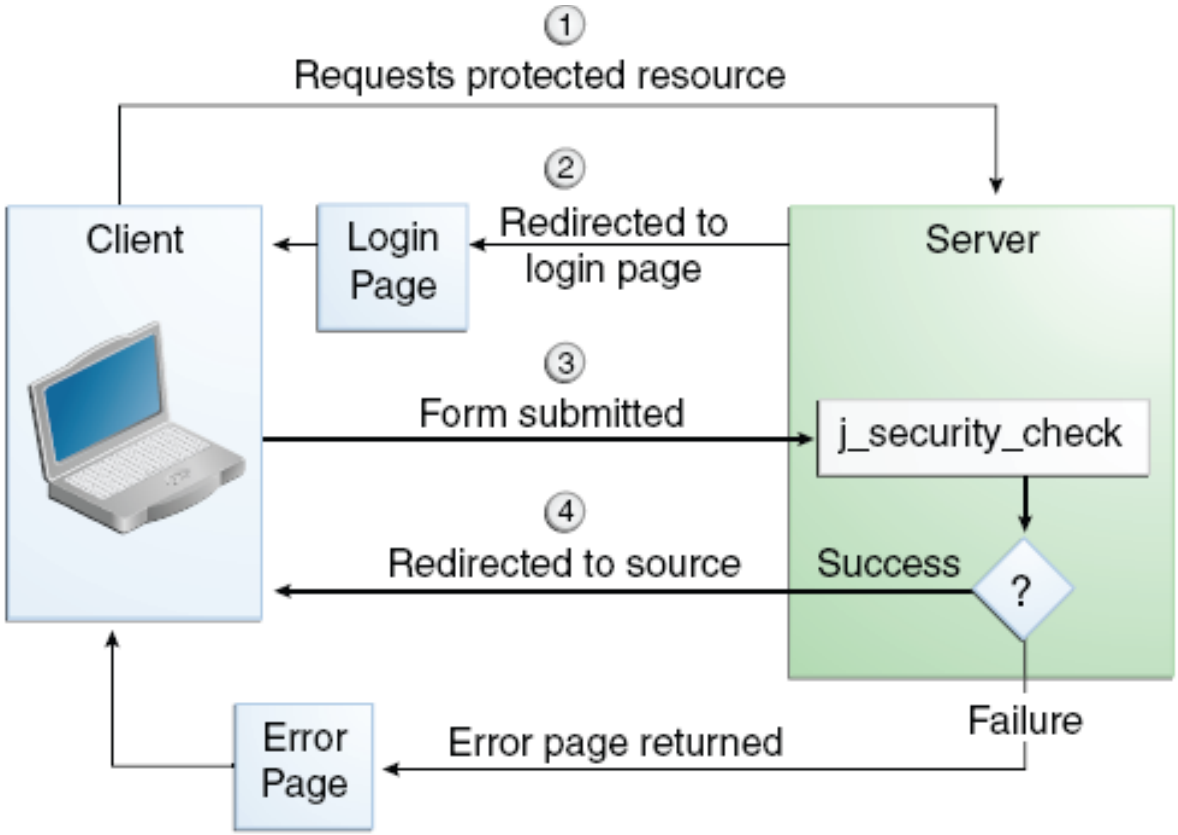


Броузер надає вікно для вводу логіну та паролю



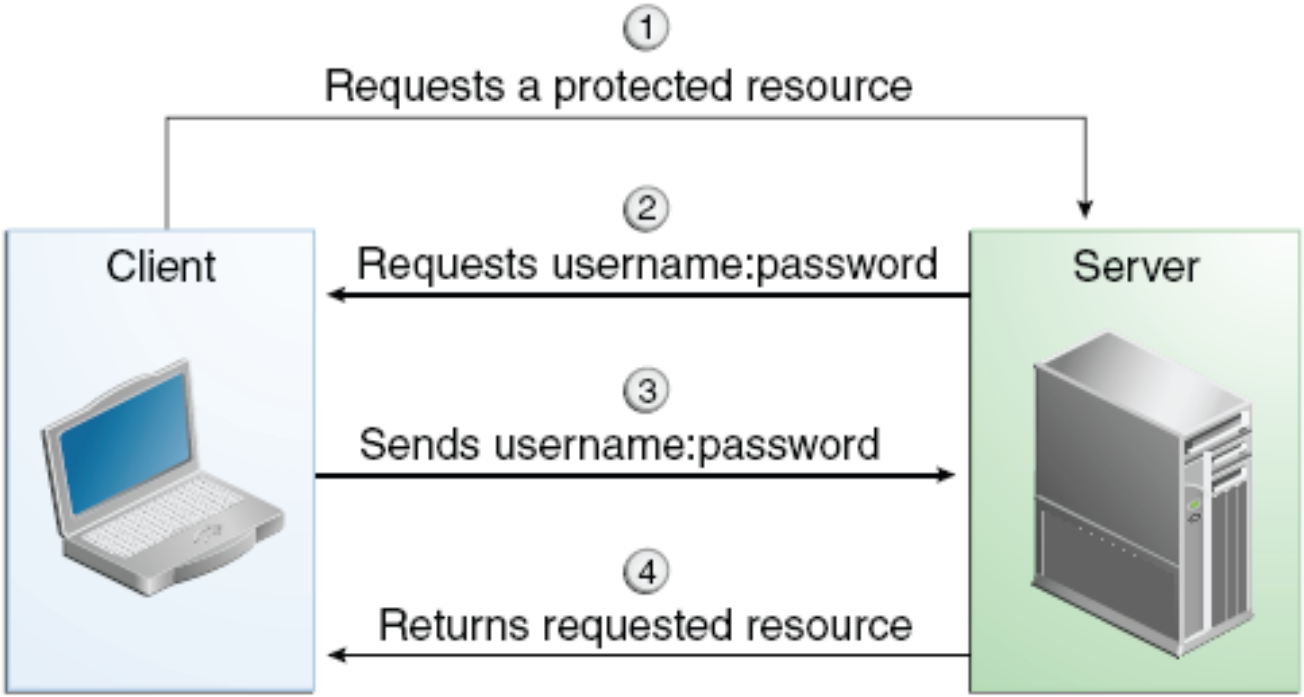
FORM

- Для вводу логіну та паролю веб-сервер перенапрявляє виклик на спеціальну сторінку логіну



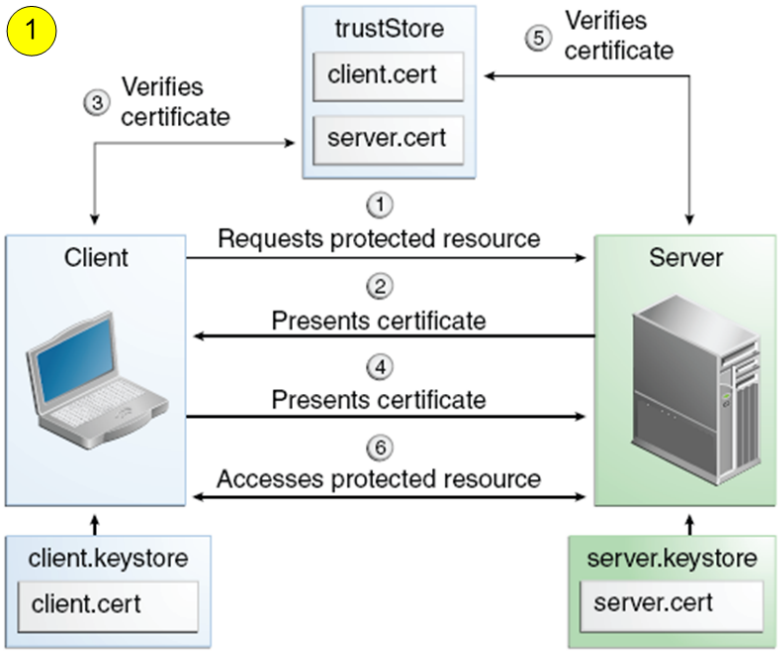
DIGEST

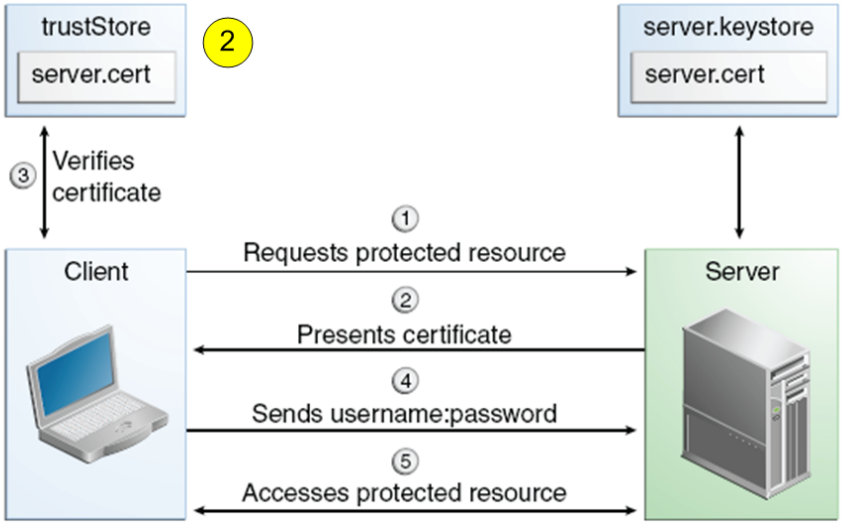
- Цей метод Basic за виключенням того, що передається не пароль, а дайждест пароля
- Підтримують не всі броузери та не всі сервери застосувань



CLIENT-CERT

- Аутентифікація на основі сертифікатів з відкритим ключем
- Два типи аутентифікації:
  1 – взаємна аутентифікація сертифікатів
  2 – взаємна аутентифікація на базі логіну/паролю





Приклад настройки логіну

Приклад для методу FORM

web.xml

- При настройці логіну вказується метод логіну та реалм безпеки

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>YOUR_REALM_NAME</realm-name>
    <form-login-config>
        <form-login-page>/login.xhtml</form-login-page>
        <form-error-page>/error.xhtml</form-error-page>
    </form-login-config>
</login-config>
```

- Для логіну по методу FORM необхідно вказати:
  - строрінку логіну
  - сторінку помилки аутентифікації

- Сторінка логіну (/login.xhtml) має містити HTML форму, подібну до наступної

```
<form method="post" action="j_security_check">
    <input type="text" name="j_username">
    <input type="password" name= "j_password">
</form>
```

у цій формі важливою є ідентифікація `j_security_check`, `j_username`, `j_password`

Декларативне завдання авторизації

- Що може бути захищеним ресурсом?
  - Сервлет
    - Доступ до сервлету Ви можете визначити у дескрипторі розгортання або у коді за допомогою анотацій
  - URL
    - Доступ ресурсів по URL визначається у дескрипторі розгортання web.xml
- Для захищених ресурсів Ви можете задати:
  - Ролі, які мають доступ
  - Настройки для транспорту (HTTP, HTTPS)

Приклад

- Декларативна настройка авторизації
- web.xml

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>wholesale</web-resource-name>
        <url-pattern>/acme/wholesale/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>PARTNER</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Визначається набір ресурсів та HTTP-методи доступу до ресурсів

Визначається роль, якій надається доступ до набору ресурсів

Задаються параметри транспорту. У даному випадку використовувати HTTPS

Програмне управління безпекою веб-застосувань

- Програмне завдання аутентифікації
  - програміст самостійно викликає метод логіну із стандартного API чи специфічного API для серверу застосувань
- Програмне завдання авторизації
  - програміст контролює доступ до ресурсів за допомогою методів `HttpServletRequest.isUserInRole`

- Програмне API

  `HttpServletRequest`
  - `getAuthType`
  - `getRemoteUser`
  - `getUserPrincipal`
  - `isUserInRole`
  - `* authenticate, login (username, password), logout`

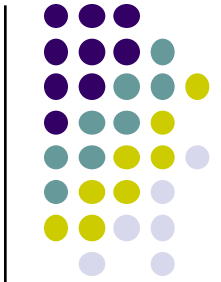
* new in Java EE 6 (Servlet 3.0)

Приклад програмного управління безпекою

```
@DeclareRoles("javaee6user")
public class LoginServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            String userName = request.getParameter("txtUserName");
            String password = request.getParameter("txtPassword");
            out.println("Before Login" + "<br><br>");
            out.println("IsUserInRole?.." + request.isUserInRole("javaee6user")+"<br>");
            out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
            out.println("getUserPrincipal?.." + request.getUserPrincipal()+"<br>");
            out.println("getAuthType?.." + request.getAuthType()+"<br><br>");
            try {
                request.login(userName, password);
            } catch(ServletException ex) {
                out.println("Login Failed with a ServletException.." + ex.getMessage());
                return;
            }
            out.println("After Login..."+"<br><br>");
            out.println("IsUserInRole?.." + request.isUserInRole("javaee6user")+"<br>");
            out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
            out.println("getUserPrincipal?.." + request.getUserPrincipal()+"<br>");
            out.println("getAuthType?.." + request.getAuthType()+"<br><br>");
            request.logout();
        } finally { out.close(); }
    }
...
}
```

Література

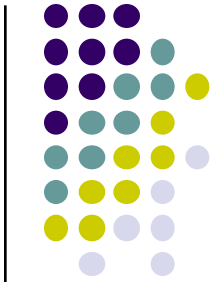
- Java EE 6 Tutorial. – 2010.
- Sun Services. FJ-310-EE5. Developing Applications for Java EE Platform. – 2008.
- Web Services Architecture. W3C Working Group Note 11 February 2004. - http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
- Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000. - http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
- Pautasso, Cesare; Zimmermann, Olaf; Leymann, Frank. RESTful Web Services vs. “Big” Web Services:Making the Right Architectural Decision, *17th International World Wide Web Conference (WWW2008)* (Beijing, China), http://www.jopera.org/docs/publications/2008/restws



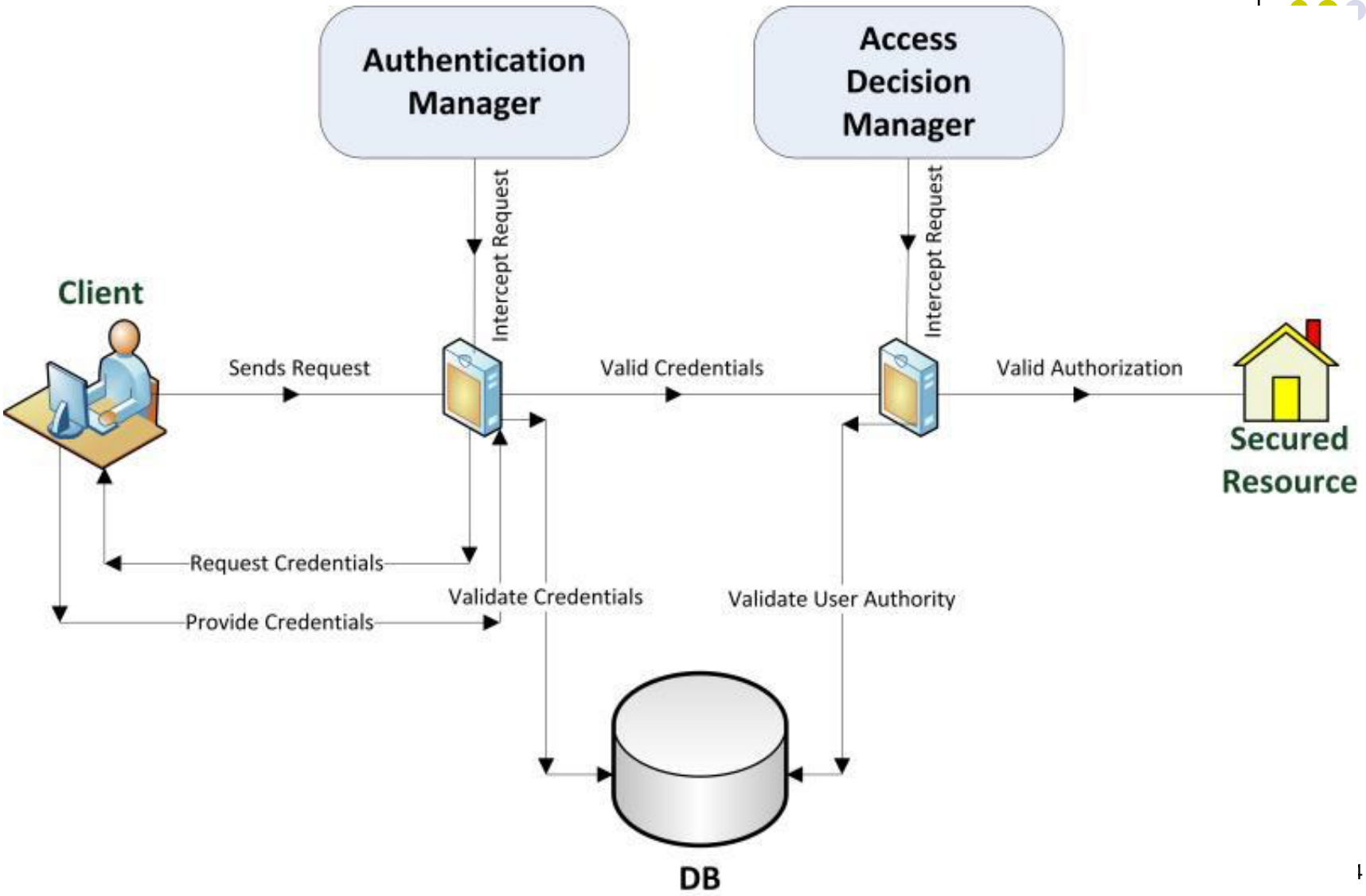
SPRING SECURITY

You can use it in:

- servlet containers
- standalone applications
- JEE environment



Big Picture


Authentication Manager
Access Decision Manager
Intercept Request
Intercept Request
Client
Sends Request
Valid Credentials
Valid Authorization
Secured Resource
Request Credentials
Provide Credentials
Validate Credentials
Validate User Authority
DB

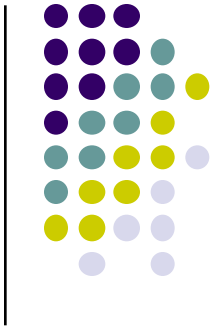
Getting started

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

}
```

Getting started with in-mem AuthenticationManager

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

   @Bean
    public AuthenticationManager authenticationManager() throws Exception {
        return new AuthenticationBuilder()
                .inMemoryAuthentication()
                        .withUser("user“)
                        .password("password")
                        .roles("USER")
                        .and()
                .and()
                .build();
    }
}
```

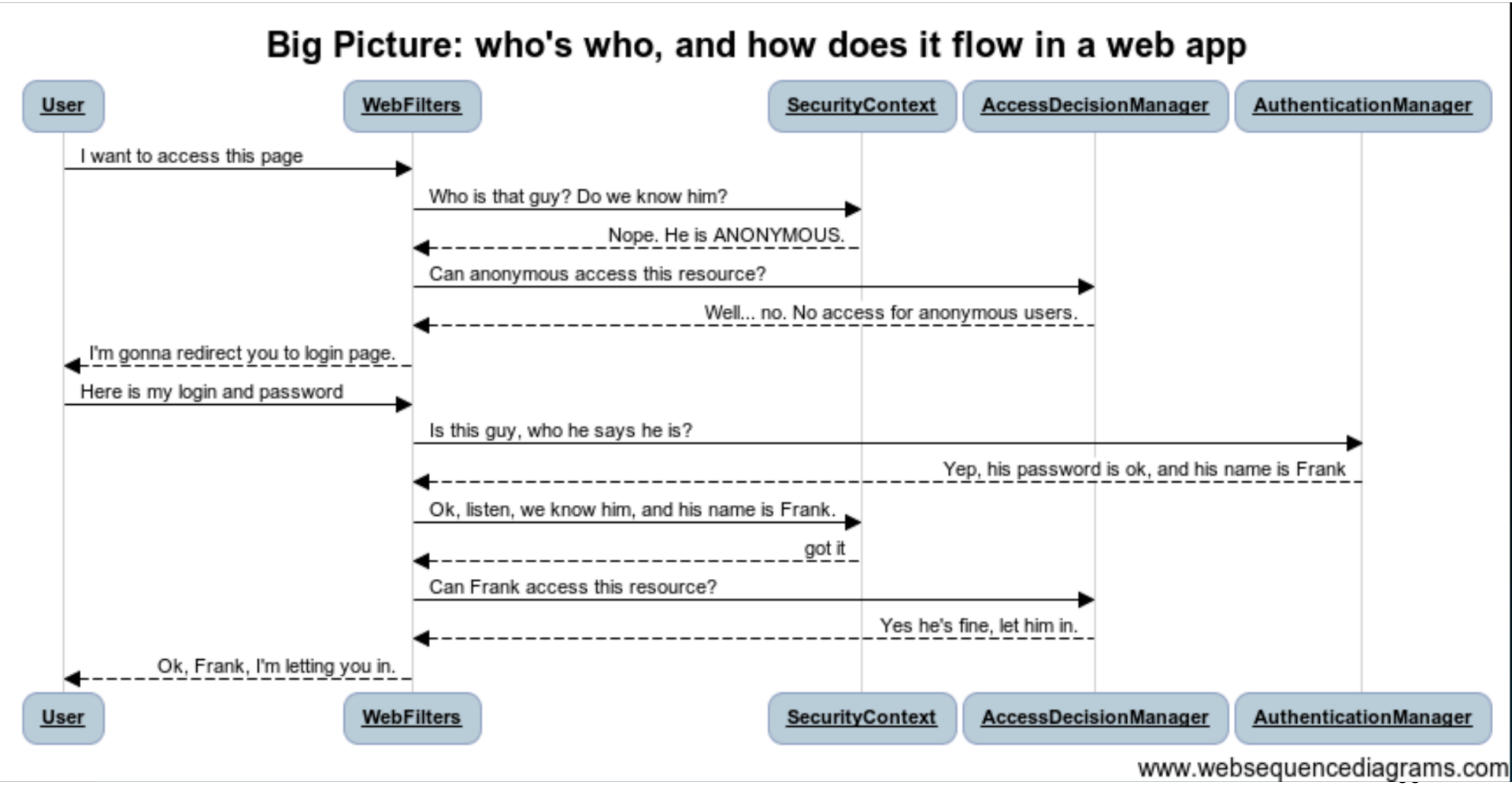
Getting started with DB AuthenticationManager



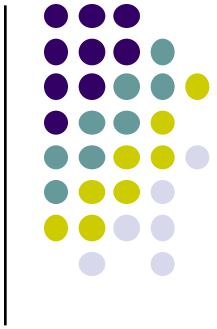
```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    DataSource dataSource;

    @Autowired
    public void configAuthentication(AuthenticationManagerBuilder auth) throws Exception {

      auth.jdbcAuthentication().dataSource(dataSource)
        .usersByUsernameQuery(
            "select username,password, enabled from users where username=?")
        .authoritiesByUsernameQuery(
            "select username, role from user_roles where username=?");
    }
```

http://www.mkyong.com/spring-security/spring-security-form-login-using-database/

Securing web resources. Big Picture

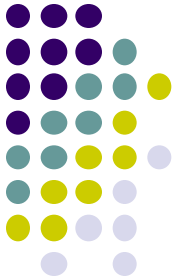


Securing Web Resources



```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
{
    public void configure(WebSecurity web) throws Exception       {
        web
        .ignoring()
        .antMatchers("/resources/**");
    }

    protected void configure(HttpSecurity http) throws Exception {
        http
        .authorizeUrls()
        .antMatchers("/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated()
   }
}
```

Let's read it

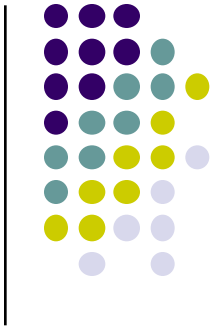


```
protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                        .anyRequest().authenticated()
                        .and()
                .formLogin()
                        .and()
                .httpBasic();
}
```

The default configuration above:

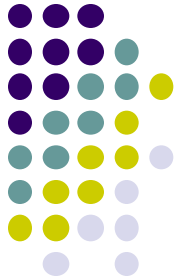
- Ensures that any request to our application requires the user to be authenticated
- Allows users to authenticate with form based login
- Allows users to authenticate with HTTP Basic authentication

Let's read this



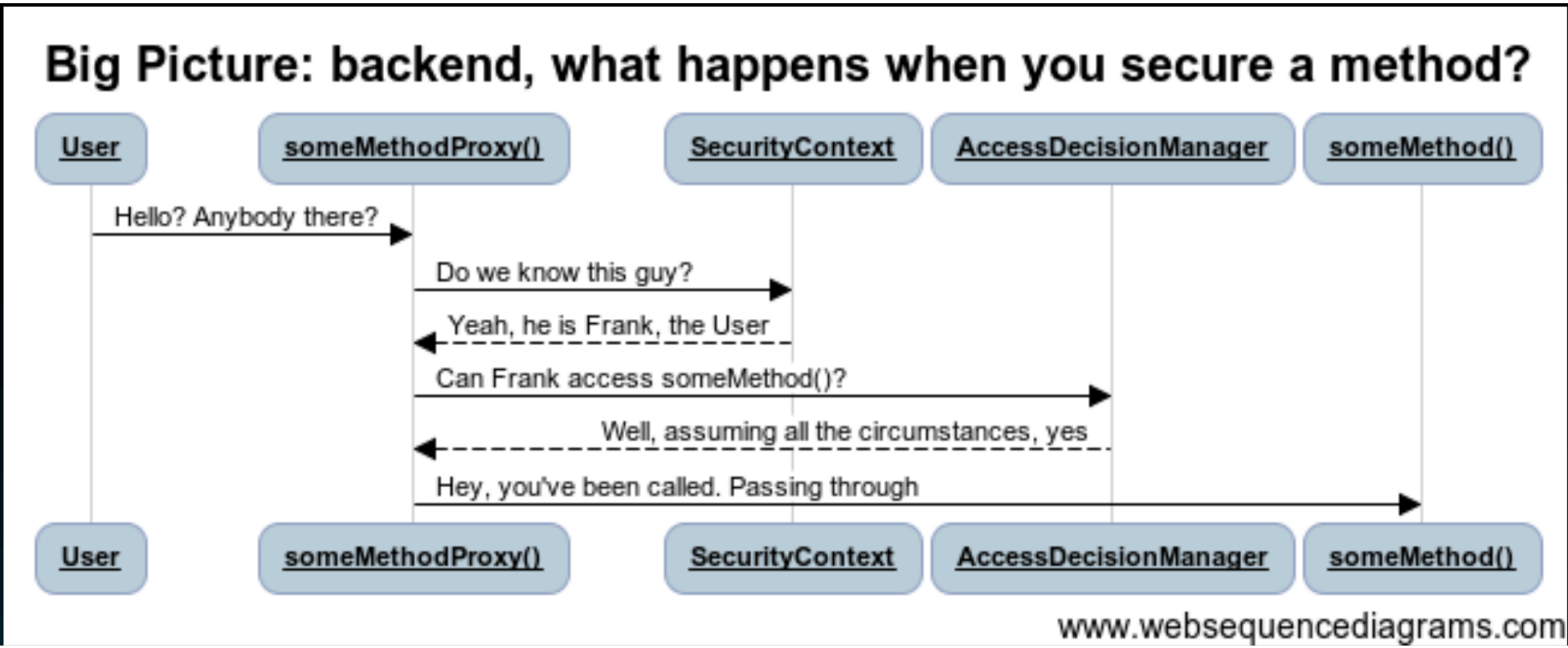
```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        .antMatchers("/resources/**", "/signup", "/about").permitAll()
        .antMatchers("/admin/**").hasRole("ADMIN")
        .antMatchers("/db/**").access("hasRole('ADMIN') and hasRole('DBA')")
        .anyRequest().authenticated()
        .and()
        // ...
        .formLogin();
}
```

Security Models for AccessDecisionManager

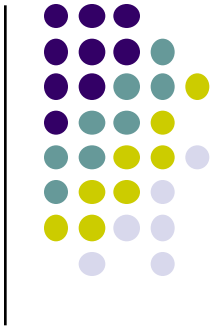


- Model based on roles
- Model based on ACLs
- Model based on business rules

Securing methods. Big Picture.



Method Security



@EnableGlobalMethodSecurity(securedEnabled = true)

```
public interface BankService {
    @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
    public Account readAccount(Long id);

    @Secured("ROLE_TELLER")
    public Account post(Account account, double amount);
}
```

@EnableGlobalMethodSecurity(prePostEnabled = true)

```
public interface BankService {
     @PreAuthorize("isAnonymous()")
     public Account readAccount(Long id);

     @PreAuthorize("hasAuthority('ROLE_TELLER')")
     public Account post(Account account, double amount);
}
```

Login form in Spring

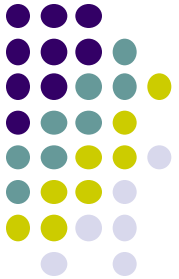
```
<c:url value="/login" var="loginUrl"/>
<form action="${loginUrl}" method="post">                    ❶
        <c:if test="${param.error != null}">                 ❷
                <p>
                        Invalid username and password.
                </p>
        </c:if>
        <c:if test="${param.logout != null}">                ❸
                <p>
                        You have been logged out.
                </p>
        </c:if>
        <p>
                <label for="username">Username</label>
                <input type="text" id="username" name="username"/>      ❹
        </p>
        <p>
                <label for="password">Password</label>
                <input type="password" id="password" name="password"/>  ❺
        </p>
        <input type="hidden"                                 ❻
                name="${_csrf.parameterName}"
                value="${_csrf.token}"/>
        <button type="submit" class="btn">Log in</button>
</form>
```


https://docs.spring.io/spring-security/site/docs/current/reference/html/jc.html#jc-form

Integration with Java EE Security

- Integrated with the following Servlet API methods
  - HttpServletRequest#getRemoteUser()
  - HttpServletRequest#getUserPrincipal()
  - HttpServletRequest#isUserInRole(java.lang.String)
  - HttpServletRequest#login(java.lang.String, java.lang.String)
  - HttpServletRequest#logout()

Demo



- https://github.com/andifalk/spring-security-workshop/tree/master/spring-security-lab3

Sources

- http://slides.com/soaserele/spring-tutorial/fullscreen#/75
- https://docs.google.com/presentation/d/1dGLJEj45yfyD1q-DeHZPECGnTR5ryu5p8_LtHC9U-Ro/edit#slide=id.i0