JSP WEB PROGRAMMING: ADVANCE

(Integrative-Generative AI Edition)

GENERATIVE AI

- Advanced Session & Security Management
- Using Expression Language (EL)
- Full Utilization of JSTL
- Performance Management and Optimization
- Project Design & Template Management

STUDENT PRICE BOOK CENTER

ai

คำนำ

ในยุคที่การพัฒนาเว็บแอปพลิเคชันต้องตอบสนองความต้องการที่ซับซ้อนและทันสมัยมากขึ้น นักพัฒนาจำเป็นต้องมีความรู้ที่ลึกซึ้งและครอบคลุมทั้งด้านเทคนิคและการออกแบบระบบเพื่อให้เว็บแอปมีประสิทธิภาพสูง ปลอดภัย และสามารถขยายขนาดได้อย่างยั่งยืน หนังสือ *JSP Web Programming: Advance* เล่มนี้จึงถูกออกแบบมาเพื่อเป็นคู่มือชั้นยอดที่ช่วยให้ผู้ที่มีพื้นฐาน JSP แล้วก้าวสู่ระดับมืออาชีพได้อย่างมั่นใจ

เนื้อหาในเล่มครอบคลุมหัวข้อที่สำคัญและจำเป็นในงานพัฒนาเว็บสมัยใหม่ เริ่มตั้งแต่การจัดการ Session และความปลอดภัยขั้นสูง (Advanced Session and Security) ซึ่งเป็นหัวใจสำคัญของระบบที่ต้องรับรองความปลอดภัยและความเป็นส่วนตัวของผู้ใช้ การเรียนรู้วิธีป้องกันการโจมตีที่เกี่ยวข้องกับ Session เช่น Session Fixation และ Session Hijacking การตั้งค่า Session Timeout อย่างเหมาะสม รวมถึงการป้องกันภัยคุกคามทางเว็บยอดนิยมอย่าง Cross-Site Scripting (XSS) และ Cross-Site Request Forgery (CSRF) ที่สามารถนำไปปรับใช้ได้ทันทีในโปรเจกต์ JSP พร้อมทั้งแนวทางการตั้งค่า HTTPS และ Server เพื่อเพิ่มความมั่นคงปลอดภัยในระดับแอปพลิเคชันและระบบโครงสร้างพื้นฐาน

ต่อมา ผู้อ่านจะได้ทำความรู้จักกับ **Expression Language (EL)** ซึ่งเป็นเทคนิคสำคัญที่ช่วยลดความซับซ้อนของโค้ด JSP โดยหลีกเลี่ยงการเขียน Scriptlet ที่ยุ่งยากและไม่ปลอดภัย หนังสือเล่มนี้จะสอนการใช้งาน EL อย่างละเอียดตั้งแต่พื้นฐานจนถึงการใช้ EL ร่วมกับ JSTL พร้อมแสดงตัวอย่างการเข้าถึง JavaBeans, Map และ List รวมถึงการใช้ Operators และ Methods ที่ช่วยให้การเขียน JSP มีความกระชับและอ่านง่ายขึ้นมาก นอกจากนี้ยังมีการสาธิตการใช้งาน EL กับ JSTL เพื่อเพิ่มความยืดหยุ่นและประสิทธิภาพของหน้าเว็บ

หัวข้อสำคัญอีกส่วนหนึ่งที่ไม่ควรพลาดคือ **JSTL (JSP Standard Tag Library)** ซึ่งเป็นเครื่องมือพื้นฐานที่ช่วยให้นักพัฒนาสามารถจัดการโฟลว์ของโปรแกรม การจัดการตัวแปร การ Query และ Update ข้อมูลในฐานข้อมูล ผ่าน SQL Tags รวมถึงการประมวลผลข้อมูล XML และการจัดรูปแบบวันที่ ตัวเลข และข้อความอย่างมืออาชีพ หนังสือเล่มนี้ให้รายละเอียดเชิงลึกของ JSTL พร้อมตัวอย่างการใช้งานที่หลากหลาย เพื่อให้ผู้อ่านสามารถนำไปประยุกต์ใช้กับโปรเจกต์จริงได้อย่างมั่นใจและมีประสิทธิภาพ

การเพิ่มประสิทธิภาพ (Performance and Optimization) ของเว็บแอปพลิเคชัน JSP เป็นอีกหนึ่งหัวข้อที่หนังสือให้ความสำคัญ บทที่ว่าด้วยการจัดการ Performance จะพาผู้อ่านไปเรียนรู้วิธีลดการใช้ Scriptlet และ Inline Java Code ที่ทำให้โค้ดช้า การใช้เทคนิค Cache ทั้งแบบเก็บผลลัพธ์ทั้งหน้าและบางส่วนของหน้า การใช้ AJAX เพื่อโหลดข้อมูลแบบไม่ต้องรีเฟรชหน้า และการตั้งค่า Server Apache Tomcat ให้เหมาะสมกับการรัน JSP รวมถึงการใช้ Connection Pooling กับ JDBC เพื่อเพิ่มประสิทธิภาพการเชื่อมต่อฐานข้อมูล ทั้งหมดนี้จะช่วยให้เว็บแอปทำงานได้รวดเร็วและรองรับผู้ใช้จำนวนมากได้อย่างมีเสถียรภาพ

ในด้านการออกแบบและจัดการหน้าตาเว็บ หนังสือเล่มนี้ยังให้ความสำคัญกับการสร้าง Template ที่มีโครงสร้างชัดเจนและดูแลรักษาง่าย ด้วยการใช้ <jsp:include> เพื่อรวมไฟล์ Header, Footer, Sidebar และการใช้ <jsp:forward> เพื่อเปลี่ยนเส้นทางการประมวลผลภายในเว็บ นอกจากนี้ยังแนะนำแนวคิด Decorator Pattern ที่ช่วยเพิ่มความยืดหยุ่นในการขยายฟังก์ชันของหน้า JSP โดยไม่ต้องแก้ไขโค้ดเดิม ซึ่งเป็นเทคนิคที่ช่วยให้นักพัฒนาสามารถออกแบบเว็บที่มีความสวยงามและทำงานได้อย่างเป็นระบบ

หนังสือ *JSP Web Programming: Advance* จึงเป็นแหล่งความรู้ที่ครบถ้วนและลึกซึ้ง เหมาะสำหรับนักพัฒนาที่ต้องการยกระดับทักษะการเขียน JSP สู่ระดับมืออาชีพ ไม่ว่าจะเป็นการจัดการความปลอดภัยขั้นสูง การใช้ Expression Language และ JSTL อย่างเต็มรูปแบบ การเพิ่มประสิทธิภาพแอปพลิเคชัน ไปจนถึงการออกแบบโครงสร้างเว็บอย่างมีมาตรฐานและประสิทธิผล ด้วยเนื้อหาที่ละเอียดพร้อมตัวอย่างประกอบชัดเจน ผู้อ่านจะได้รับทั้งความรู้และทักษะที่พร้อมนำไปใช้ในโปรเจกต์จริง ทั้งในองค์กรและงานส่วนตัว เพื่อก้าวสู่การเป็นนักพัฒนา JSP มืออาชีพอย่างแท้จริง

ขอให้ผู้อ่านทุกท่านสนุกกับการเรียนรู้และพัฒนาทักษะ JSP ผ่านหนังสือเล่มนี้ และประสบความสำเร็จในเส้นทางการพัฒนาเว็บแอปพลิเคชันอย่างมั่นคงและยั่งยืน.

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 10

การจัดการ Session และ Security ขั้นสูง

(Advanced Session and Security)

เนื้อหา

- การจัดการ Session และ Security ขั้นสูง
- การจัดการ Session และ Security ขั้นสูง (เชิงลึก)
- เทคนิคการจัดการ Session Fixation และ Session Hijacking
- การตั้งค่า Session Timeout และการจัดการ Session หมดอายุ
- การป้องกัน Cross-Site Scripting (XSS) ด้วยการ Escape Output
- การป้องกัน Cross-Site Request Forgery (CSRF) ใน JSP Web Application
- การใช้ HTTPS กับ JSP และการตั้งค่า Server

บทนำ – บทที่ 10: การจัดการ Session และ Security ขั้นสูง

เมื่อเว็บแอปพลิเคชันเติบโตขึ้นและมีผู้ใช้งานจำนวนมาก ความปลอดภัย (Security) กลายเป็นประเด็นสำคัญที่ไม่สามารถมองข้ามได้ โดยเฉพาะในระบบที่มีการจัดการผู้ใช้ การล็อกอิน หรือข้อมูลสำคัญ การป้องกันช่องโหว่ด้าน Session และความปลอดภัยจึงเป็นพื้นฐานสำคัญที่ทุกนักพัฒนา JSP ควรเข้าใจอย่างลึกซึ้ง บทที่ 10 นี้จะพาผู้อ่านเจาะลึกการจัดการ Session และแนวทางป้องกันความเสี่ยงด้านความปลอดภัยในระดับที่สูงขึ้น

บทเรียนเริ่มต้นด้วยการอธิบาย **ภัยคุกคามยอดนิยมที่เกี่ยวข้องกับ Session** ได้แก่ *Session Fixation* และ *Session Hijacking* ซึ่งเป็นเทคนิคที่ผู้ไม่หวังดีใช้เพื่อแอบอ้างสิทธิ์หรือขโมย Session ของผู้ใช้ที่ล็อกอินแล้ว ผู้อ่านจะได้เข้าใจถึงรูปแบบการโจมตีและวิธีการป้องกัน เช่น การสร้าง Session ใหม่หลัง Login หรือการตรวจสอบ IP และ User-Agent

จากนั้นผู้อ่านจะได้เรียนรู้วิธี **ตั้งค่า Session Timeout** อย่างเหมาะสม เพื่อให้ Session หมดอายุอัตโนมัติหลังไม่มีการใช้งาน พร้อมแนวทางการจัดการเมื่อ Session หมดอายุ เช่น การ redirect ไปยังหน้า Login หรือแสดงข้อความหมดเวลาอย่างปลอดภัย รวมถึงการตั้งค่าใน web.xml และโค้ด JSP/Servlet

หนึ่งในภัยคุกคามที่พบบ่อยมากคือ **Cross-Site Scripting (XSS)** ซึ่งเกิดจากการที่ผู้ใช้สามารถป้อน JavaScript หรือ HTML อันตรายลงในระบบ แล้วทำให้ถูกแสดงผลในหน้าเว็บของผู้ใช้คนอื่น บทนี้จะแนะนำการ *Escape Output* อย่างถูกต้อง เช่น การใช้ <c:out> ของ JSTL หรือการเขียนฟังก์ชัน escape ด้วยตนเอง เพื่อป้องกันช่องโหว่นี้

ต่อมา ผู้อ่านจะได้รู้จักกับ **การโจมตีแบบ Cross-Site Request Forgery (CSRF)** ซึ่งเป็นการปลอมคำร้องขอจากผู้ใช้ที่เข้าสู่ระบบอยู่ โดยไม่รู้ตัว เทคนิคในการป้องกันคือการใช้ *CSRF Token* ซึ่งจะฝังไว้ในฟอร์มทุกครั้ง และตรวจสอบความถูกต้องเมื่อรับค่ากลับมาที่เซิร์ฟเวอร์ ซึ่งบทนี้จะอธิบายและมีตัวอย่างประกอบอย่างชัดเจน

นอกจากนี้ยังมีหัวข้อ **การใช้ HTTPS** เพื่อเข้ารหัสข้อมูลที่ส่งผ่านเครือข่ายระหว่างผู้ใช้กับเซิร์ฟเวอร์ ป้องกันไม่ให้ข้อมูลถูกดักจับหรือแก้ไขได้ บทเรียนจะแนะนำวิธีตั้งค่าให้ JSP ทำงานผ่าน HTTPS พร้อมการตั้งค่า SSL/TLS บน Server เช่น Apache Tomcat เพื่อเสริมความปลอดภัยทั้งในระดับแอปพลิเคชันและโครงสร้างพื้นฐาน

บทนี้จึงเหมาะสำหรับผู้ที่ต้องการยกระดับความเข้าใจเรื่อง Session และ Security ในระดับองค์กร โดยเน้นทั้งแนวคิด ภัยคุกคาม และแนวทางปฏิบัติที่ปลอดภัยในระบบจริง พร้อมตัวอย่างและเทคนิคที่สามารถนำไปใช้ได้ทันทีในโปรเจกต์ JSP ที่ต้องรองรับผู้ใช้งานจริง เมื่อจบบทนี้ ผู้อ่านจะมีความพร้อมในการวางระบบ JSP ที่ปลอดภัยมากขึ้น รู้จักวิธีป้องกันการโจมตีจากผู้ไม่หวังดี และสามารถจัดการ Session อย่างมีประสิทธิภาพ ทั้งในด้านความปลอดภัย การจัดการทรัพยากร และการปกป้องข้อมูลผู้ใช้.

การจัดการ Session และ Security ขั้นสูง

---

1. เทคนิคการจัดการ Session Fixation และ Session Hijacking

- **Session Fixation** คือการโจมตีที่แฮกเกอร์ตั้งค่า Session ID ล่วงหน้า แล้วหลอกให้ผู้ใช้ใช้ Session ID นั้น จากนั้นแฮกเกอร์จะใช้ Session ID เดียวกันเข้าถึงระบบ
- **แนวทางป้องกัน**
    - สร้าง Session ใหม่ (Regenerate Session ID) หลังจากผู้ใช้ล็อกอินสำเร็จ เช่น request.getSession().invalidate() แล้ว request.getSession(true)
    - ตั้งค่าคุณสมบัติ HttpOnly กับคุกกี้ Session เพื่อป้องกันการขโมยผ่าน JavaScript
    - ใช้ Secure Cookie (เฉพาะ HTTPS เท่านั้น)
    - กำหนดค่า SameSite Cookie (Lax หรือ Strict) เพื่อป้องกันการส่งคุกกี้จากโดเมนอื่น
- **Session Hijacking** คือการขโมย Session ID เพื่อเข้าใช้งานระบบแทนผู้ใช้
    - ป้องกันด้วย HTTPS
    - ตั้งค่า Timeout Session ให้เหมาะสม
    - ตรวจสอบ IP Address หรือ User-Agent ใน Session

---

2. การตั้งค่า Session Timeout และการจัดการ Session หมดอายุ

- **Session Timeout** คือระยะเวลาที่ Session จะหมดอายุถ้าไม่มีการใช้งาน
- วิธีตั้งค่าบน web.xml:

```
<session-config>
    <session-timeout>30</session-timeout> <!-- หน่วยเป็นนาที -->
</session-config>
```

- หรือในโค้ด JSP/Servlet:

```
request.getSession().setMaxInactiveInterval(1800); // 1800 วินาที = 30 นาที
```

- เมื่อ Session หมดอายุ ให้ Redirect ไปหน้า Login หรือแจ้งเตือนผู้ใช้

---

3. การป้องกัน Cross-Site Scripting (XSS) ด้วยการ Escape Output

- XSS เกิดจากการที่ข้อมูลที่ผู้ใช้ป้อนถูกแสดงผลโดยไม่ถูกกรอง ทำให้สามารถแทรกสคริปต์อันตรายเข้าไปได้
- วิธีป้องกัน
    - **Escape Output** ก่อนแสดงบนหน้า JSP เช่น ใช้ JSTL <c:out>:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:out value="${userInput}" />
```

- หรือใช้ฟังก์ชัน Java ที่แปลงอักขระพิเศษ เช่น <, >, & เป็น Entity
- หลีกเลี่ยงการใช้ <%= userInput %> โดยตรงโดยไม่กรอง

---

4. การป้องกัน Cross-Site Request Forgery (CSRF) ด้วย Token

- CSRF คือการโจมตีโดยใช้สิทธิ์ของผู้ใช้ผ่านฟอร์มที่ไม่ได้รับอนุญาต
- แนวทางป้องกัน
    - สร้าง **CSRF Token** (random string) เมื่อโหลดฟอร์ม และเก็บใน Session
    - ส่ง Token นี้ใน hidden form field หรือ header ทุกครั้งที่ส่งฟอร์ม
    - เมื่อ Server รับคำขอ ตรวจสอบว่า Token ที่ส่งมาตรงกับที่เก็บใน Session หรือไม่
    - ตัวอย่างโค้ดใน JSP:

```
<%
    String csrfToken = (String) session.getAttribute("csrfToken");
    if(csrfToken == null) {
        csrfToken = java.util.UUID.randomUUID().toString();
        session.setAttribute("csrfToken", csrfToken);
    }
%>
<form method="post" action="process.jsp">
    <input type="hidden" name="csrfToken" value="<%= csrfToken %>" />
    <!-- ฟอร์มอื่นๆ -->
```

```
</form>
```

- ใน process.jsp ให้ตรวจสอบ Token:

```
String sessionToken = (String) session.getAttribute("csrfToken");
String requestToken = request.getParameter("csrfToken");

if(sessionToken == null || !sessionToken.equals(requestToken)) {
    // แสดง error หรือปฏิเสธคำขอ
}
```

---

5. การใช้ HTTPS กับ JSP และการตั้งค่า Server

- HTTPS ช่วยป้องกันการดักจับข้อมูลระหว่างทาง และป้องกัน Session Hijacking
- ขั้นตอนใช้งาน HTTPS
    - ขอใบรับรอง SSL Certificate จาก CA เช่น Let's Encrypt หรือซื้อจากผู้ให้บริการ
    - ติดตั้ง SSL Certificate บน Web Server หรือ Application Server (เช่น Tomcat, Apache HTTP Server)
    - ตั้งค่า Server ให้บังคับ Redirect จาก HTTP เป็น HTTPS
    - ตัวอย่างการตั้งค่า HTTPS ใน Tomcat:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true">
    <SSLHostConfig>
        <Certificate certificateKeystoreFile="conf/keystore.jks"
                     type="RSA" />
    </SSLHostConfig>
</Connector>
```

- ใน JSP/Servlet ควรตรวจสอบว่า Request มาจาก HTTPS เพื่อ Redirect หรือแสดงข้อมูลอย่างเหมาะสม

การจัดการ Session และ Security ขั้นสูง (เชิงลึก)

---

1. เทคนิคการจัดการ Session Fixation และ Session Hijacking

Session Fixation

- **หลักการโจมตี**: ผู้โจมตีเตรียม Session ID ไว้ล่วงหน้า (เช่น ส่งลิงก์ให้เหยื่อที่มี Session ID นั้น) และเมื่อตัวเหยื่อล็อกอิน Session ID นี้จะถูกนำไปใช้ต่อ ทำให้ผู้โจมตีสามารถเข้าถึง Session ของเหยื่อได้

- **แนวทางป้องกันใน JSP/Servlet**:
    - หลังจากที่ผู้ใช้ล็อกอินสำเร็จ **ควรทำการเปลี่ยน Session ID ใหม่ทันที**
    - วิธีการ:

```
HttpSession oldSession = request.getSession(false);
if (oldSession != null) {
    oldSession.invalidate(); // ทำลาย session เก่า
}
HttpSession newSession = request.getSession(true); // สร้าง session ใหม่
// เซ็ตค่า attribute ใหม่ใน session ใหม่ถ้าจำเป็น
```

- การเปลี่ยน Session ID ช่วยให้ Session ที่แฮกเกอร์รู้จักไม่สามารถใช้ได้อีกต่อไป
- ควรตั้งค่าคุกกี้ session เป็น HttpOnly และ Secure เพื่อป้องกันการขโมย session ผ่านทางสคริปต์และทำงานเฉพาะ HTTPS

Session Hijacking

- การขโมย Session ID โดยแฮกเกอร์ผ่านวิธีต่าง ๆ เช่น Packet Sniffing, Cross-Site Scripting (XSS), หรือ Session Fixation
- **แนวทางป้องกัน**:
    - ใช้ HTTPS ตลอดเวลาในการสื่อสาร (ป้องกันการถูกดักข้อมูลในเครือข่าย)
    - กำหนด HttpOnly และ Secure flags ให้กับคุกกี้ session
    - กำหนด Session Timeout ที่เหมาะสม
    - ตรวจสอบ IP Address หรือ User-Agent ของ client ทุกครั้งใน Session (สามารถเปรียบเทียบข้อมูลนี้กับที่เก็บใน Session) หากผิดปกติให้ยกเลิก Session

---

2. การตั้งค่า Session Timeout และการจัดการ Session หมดอายุ

- **Session Timeout** คือเวลาที่ Session จะหมดอายุหลังจากไม่มีการใช้งาน
- การตั้งค่า Timeout:
    - **ใน web.xml**
    - <session-config>
    - <session-timeout>15</session-timeout> <!-- 15 นาที -->
    - </session-config>
    - **ในโค้ด Servlet หรือ JSP**
    - HttpSession session = request.getSession();
    - session.setMaxInactiveInterval(15 * 60); // วินาที
- การจัดการเมื่อ Session หมดอายุ:
    - ตรวจสอบ Session ว่ายังมีอยู่หรือไม่ในทุกหน้าที่ต้องการความปลอดภัย

- หาก Session หมดอายุหรือไม่มี Session ให้ Redirect ไปหน้า Login
- ตัวอย่างโค้ดเช็ค Session:
- HttpSession session = request.getSession(false);
- if (session == null || session.getAttribute("user") == null) {
- response.sendRedirect("login.jsp");
- return;
- }

---

3. การป้องกัน Cross-Site Scripting (XSS) ด้วยการ Escape Output

- XSS คือการแทรกโค้ด JavaScript หรือโค้ดอันตรายลงในหน้าเว็บผ่านข้อมูลป้อนเข้าจากผู้ใช้
- **วิธีป้องกันหลัก**: ไม่แสดงข้อมูลที่รับมาจากผู้ใช้โดยตรง แต่ต้องแปลงอักขระพิเศษให้ปลอดภัย (escape)
- ใน JSP มีเครื่องมือช่วยเช่น JSTL <c:out>
- <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
- <c:out value="${param.input}" />
- ถ้าไม่ใช้ JSTL ต้องเขียนฟังก์ชัน escape อักขระเอง เช่น:
- public static String escapeHtml(String input) {
- if (input == null) return "";
- return input.replace("&", "&")
- .replace("<", "<")
- .replace(">", ">")
- .replace("\"", """)
- .replace("'", "'");
- }
- นอกจากนี้ควรทำ sanitize input ที่ฝั่ง Server และ Client

---

4. การป้องกัน Cross-Site Request Forgery (CSRF) ด้วย Token

- CSRF คือการหลอกให้ผู้ใช้ที่ล็อกอินในระบบ ส่งคำขอที่ไม่ได้ตั้งใจไปยังเซิร์ฟเวอร์ (เช่น การโอนเงิน, เปลี่ยนรหัสผ่าน)
- **แนวทางป้องกันด้วย CSRF Token**:
  - สร้าง Token แบบสุ่มและเก็บไว้ใน Session เมื่อแสดงฟอร์ม
  - ส่ง Token นี้พร้อมข้อมูลฟอร์มใน hidden field
  - เมื่อรับคำขอ POST ตรวจสอบว่า Token ที่ส่งมาตรงกับที่เก็บใน Session
- ตัวอย่างการสร้างและตรวจสอบ Token:

- **สร้าง Token ใน JSP:**
- <%
- String csrfToken = (String) session.getAttribute("csrfToken");
- if (csrfToken == null) {
- csrfToken = java.util.UUID.randomUUID().toString();
- session.setAttribute("csrfToken", csrfToken);
- }
- %>
- <form method="post" action="process.jsp">
- <input type="hidden" name="csrfToken" value="<%= csrfToken %>" />
- <!-- ฟอร์มอื่นๆ -->
- </form>
- **ตรวจสอบ Token ใน Servlet/JSP:**
- String sessionToken = (String) session.getAttribute("csrfToken");
- String requestToken = request.getParameter("csrfToken");
-
- if (sessionToken == null || !sessionToken.equals(requestToken)) {
- response.sendError(HttpServletResponse.SC_FORBIDDEN, "Invalid CSRF token");
- return;
- }

- ในระบบจริง ควรสร้าง Token ใหม่หลังจากใช้แล้ว (One-time token)

---

5. การใช้ HTTPS กับ JSP และการตั้งค่า Server

- HTTPS เป็นสิ่งจำเป็นสำหรับความปลอดภัยของข้อมูล และป้องกัน Session Hijacking และ Man-in-the-Middle
- **การตั้งค่า HTTPS บน Server (เช่น Tomcat)**:
  - สร้างหรือขอ SSL Certificate จาก CA ที่เชื่อถือได้ (Let's Encrypt ฟรี, หรือซื้อจากผู้ให้บริการ)
  - นำ Certificate ใส่ใน Keystore (.jks file)
  - แก้ไขไฟล์ server.xml ใน Tomcat เพิ่ม Connector สำหรับ HTTPS เช่น:
  - <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
  - maxThreads="150" SSLEnabled="true" scheme="https" secure="true"

- keystoreFile="conf/keystore.jks"
  keystorePass="your_keystore_password"
- clientAuth="false" sslProtocol="TLS" />

- **บังคับ Redirect HTTP เป็น HTTPS**
  - บน Tomcat อาจทำผ่าน Filter หรือใช้ Apache HTTP Server/NGINX ด้านหน้า Redirect
  - ตัวอย่าง Filter ใน JSP/Servlet:
  - if (!request.isSecure()) {
  - String redirectURL = "https://" + request.getServerName() + request.getRequestURI();
  - if (request.getQueryString() != null) {
  - redirectURL += "?" + request.getQueryString();
  - }
  - response.sendRedirect(redirectURL);
  - return;
  - }
- **ตั้งค่า Cookie Secure และ HttpOnly**
  - ใน Tomcat, ตั้งค่าใน context.xml หรือใช้โค้ด Java:
  - Cookie sessionCookie = new Cookie("JSESSIONID", session.getId());
  - sessionCookie.setSecure(true);
  - sessionCookie.setHttpOnly(true);
  - response.addCookie(sessionCookie);
- HTTPS ยังช่วยให้ Browser สามารถใช้คุณสมบัติ SameSite Strict/Lax ป้องกัน CSRF ได้ดีขึ้น

สรุปเชิงลึก

หัวข้อ	เทคนิคและแนวทางเชิงลึก
Session Fixation	เปลี่ยน Session ID หลังล็อกอิน, ใช้ HttpOnly, Secure cookie, ตรวจสอบ IP/User-Agent
Session Hijacking	ใช้ HTTPS, Timeout session, ตรวจสอบพฤติกรรม Session
Session Timeout	ตั้งเวลาใน web.xml หรือโค้ด, ตรวจสอบ Session ทุกครั้งก่อนเข้าหน้า secured

หัวข้อ	เทคนิคและแนวทางเชิงลึก
XSS	Escape Output ด้วย JSTL <c:out>, sanitize input/output, หลีกเลี่ยง echo ตรงแบบ raw
CSRF	สร้างและตรวจสอบ CSRF Token ในฟอร์มและ Server, ใช้ Token แบบสุ่มและเปลี่ยนทุกครั้งที่ใช้
HTTPS	ตั้งค่า SSL Cert ใน Server, Redirect HTTP->HTTPS, ตั้ง Secure & HttpOnly cookie, บังคับใช้งาน HTTPS ตลอด

เทคนิคการจัดการ Session Fixation และ Session Hijacking

1. Session Fixation

Session Fixation คือ การโจมตีที่แฮกเกอร์จะทำให้เหยื่อใช้ Session ID ที่แฮกเกอร์รู้หรือกำหนดไว้ล่วงหน้า หลังจากที่เหยื่อล็อกอินเข้าสู่ระบบ แฮกเกอร์ก็สามารถใช้ Session ID นั้นเข้าถึงระบบแทนเหยื่อได้

วิธีโจมตี (Session Fixation) แบบง่าย ๆ

- แฮกเกอร์ส่งลิงก์ที่มี Session ID (หรือ Cookie) ที่กำหนดไว้ เช่น
  http://example.com/app;jsessionid=12345
- เหยื่อคลิกลิงก์นี้และเข้าสู่ระบบ
- ระบบยังคงใช้ Session ID เดิม (12345)
- แฮกเกอร์ใช้ Session ID นี้เข้าถึงระบบแทนเหยื่อ

แนวทางป้องกัน

- **สร้าง Session ใหม่ (Regenerate Session ID) หลังจากล็อกอินสำเร็จ**
  - ปิด session เก่า แล้วสร้าง session ใหม่ทันที
  - ป้องกันไม่ให้ Session ID ที่แฮกเกอร์รู้ถูกใช้งานหลังล็อกอิน
- ตัวอย่างโค้ด JSP/Servlet:

```
// ก่อนดำเนินการล็อกอิน
HttpSession oldSession = request.getSession(false);
if (oldSession != null) {
    oldSession.invalidate();  // ปิด session เดิม
}

// สร้าง session ใหม่หลังล็อกอิน
HttpSession newSession = request.getSession(true);
```

newSession.setAttribute("user", loggedInUser);

- **ตั้งค่า Cookie ให้มี HttpOnly และ Secure flag**
    - HttpOnly ป้องกันไม่ให้ JavaScript เข้าถึง Session Cookie
    - Secure ทำให้ Cookie ถูกส่งผ่าน HTTPS เท่านั้น
- วิธีตั้งค่าคุกกี้ Session ให้ Secure + HttpOnly:

```
Cookie sessionCookie = new Cookie("JSESSIONID", newSession.getId());
sessionCookie.setHttpOnly(true);
sessionCookie.setSecure(true);
response.addCookie(sessionCookie);
```

- **ตรวจสอบ IP หรือ User-Agent** (Optional)
    - เก็บ IP และ User-Agent ตอนสร้าง Session
    - เปรียบเทียบทุก request ถ้ามีความผิดปกติให้ยกเลิก Session

---

2. Session Hijacking

Session Hijacking คือการขโมย Session ID ของผู้ใช้โดยแฮกเกอร์ เพื่อเข้าใช้ระบบแทนผู้ใช้โดยไม่ได้รับอนุญาต

วิธีโจมตีที่พบบ่อย

- ดักจับ Session ID ในการสื่อสารแบบ HTTP (ไม่เข้ารหัส)
- ขโมย Session ID จากช่องโหว่ XSS
- ใช้ Session Fixation

แนวทางป้องกัน

- **ใช้ HTTPS ตลอดการสื่อสาร**
    - ป้องกันการดักจับข้อมูลจาก network
- **ตั้งค่า Cookie Secure และ HttpOnly** (เหมือนข้อ Session Fixation)
    - ป้องกันไม่ให้ cookie ถูกอ่านโดย JavaScript หรือส่งผ่าน HTTP
- **ตั้งค่า Session Timeout ให้เหมาะสม**
    - จำกัดเวลาการใช้งาน Session หากไม่ใช้งานนานเกินไปให้หมดอายุ
- **ตรวจสอบพฤติกรรมผิดปกติใน Session**
    - ตรวจสอบ IP Address หรือ User-Agent ว่าตรงกับที่สร้าง Session หรือไม่
    - หากไม่ตรงให้ invalidate session
- **เพิ่มการตรวจสอบ Two-Factor Authentication (2FA)**
    - เพิ่มชั้นความปลอดภัยนอกเหนือจาก Session ID

ตัวอย่างโค้ดตรวจสอบ IP Address ใน Session

```
HttpSession session = request.getSession(false);
if (session != null) {
    String sessionIP = (String) session.getAttribute("clientIP");
    String currentIP = request.getRemoteAddr();
    if (sessionIP == null) {
        session.setAttribute("clientIP", currentIP);
    } else if (!sessionIP.equals(currentIP)) {
        session.invalidate(); // ยกเลิก session หาก IP เปลี่ยน
        response.sendRedirect("login.jsp");
        return;
    }
}
```

สรุปแนวทางที่สำคัญ

เทคนิค	รายละเอียด
Regenerate Session ID	สร้าง Session ใหม่หลังล็อกอิน เพื่อลดความเสี่ยง Session Fixation
ตั้งค่า HttpOnly Cookie	ป้องกัน JavaScript เข้าถึง Session Cookie
ตั้งค่า Secure Cookie	ส่ง Cookie เฉพาะผ่าน HTTPS
ใช้ HTTPS ตลอดการเชื่อมต่อ	ป้องกันการดักจับข้อมูล Session
ตั้ง Session Timeout	จำกัดเวลาการใช้งาน Session
ตรวจสอบ IP/User-Agent	เปรียบเทียบข้อมูล client กับที่เก็บใน Session

---

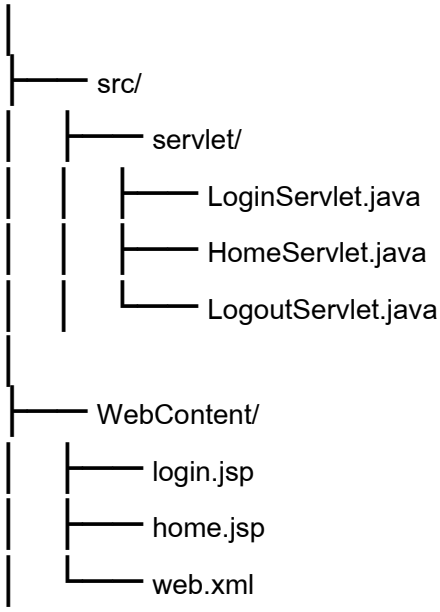
ด้านล่างนี้เป็นตัวอย่าง **3 โปรแกรมพื้นฐาน** และ **3 โปรแกรมแนวประยุกต์** ที่แสดงการจัดการ **Session Fixation และ Session Hijacking** ด้วย JSP + Servlet พร้อมโครงสร้าง, คำอธิบายโค้ด และผลการรัน

ตัวอย่างพื้นฐาน 3 โปรแกรม

เน้นการป้องกัน Session Fixation และตรวจสอบ Session Hijacking (IP Check)

โครงสร้างโปรเจกต์พื้นฐาน

SessionSecurityBasic/



1. LoginServlet.java

```
package servlet;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // สมมุติฐานตรวจสอบ user/pass แบบง่าย ๆ
        if ("user".equals(username) && "pass".equals(password)) {
            // ป้องกัน Session Fixation: ลบ session เก่าแล้วสร้างใหม่
            HttpSession oldSession = request.getSession(false);
            if (oldSession != null) {
                oldSession.invalidate();
            }
            HttpSession newSession = request.getSession(true);
            newSession.setAttribute("username", username);
```

```
            // เก็บ IP Address สำหรับตรวจสอบ Session Hijacking
            String clientIP = request.getRemoteAddr();
            newSession.setAttribute("clientIP", clientIP);

            // ตั้งเวลา session timeout 15 นาที
            newSession.setMaxInactiveInterval(15 * 60);

            response.sendRedirect("home");
        } else {
            request.setAttribute("errorMessage", "Invalid username or password");
            request.getRequestDispatcher("login.jsp").forward(request, response);
        }
    }
}
```

---

2. HomeServlet.java

```
package servlet;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class HomeServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        HttpSession session = request.getSession(false);

        if (session == null || session.getAttribute("username") == null) {
            response.sendRedirect("login.jsp");
            return;
        }

        // ตรวจสอบ IP Address เพื่อป้องกัน Session Hijacking
```

```
        String sessionIP = (String) session.getAttribute("clientIP");
        String currentIP = request.getRemoteAddr();
        if (!currentIP.equals(sessionIP)) {
            session.invalidate();
            response.sendRedirect("login.jsp");
            return;
        }

        request.getRequestDispatcher("home.jsp").forward(request, response);
    }
}
```

---

3. LogoutServlet.java

```
package servlet;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class LogoutServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        HttpSession session = request.getSession(false);
        if (session != null) {
            session.invalidate();
        }
        response.sendRedirect("login.jsp");
    }
}
```

---

4. login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<head><title>Login</title></head>
```

```
<body>
<h2>Login</h2>
<form method="post" action="login">
    Username: <input type="text" name="username" required /><br/>
    Password: <input type="password" name="password" required /><br/>
    <input type="submit" value="Login" />
</form>
<c:if test="${not empty errorMessage}">
    <p style="color:red;">${errorMessage}</p>
</c:if>
</body>
</html>
```

---

5. home.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" session="true" %>
<html>
<head><title>Home</title></head>
<body>
<h2>Welcome, <%= session.getAttribute("username") %>!</h2>
<a href="logout">Logout</a>
</body>
</html>
```

---

6. web.xml

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">
  <servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>servlet.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/login</url-pattern>
  </servlet-mapping>
```

```
  <servlet>
    <servlet-name>HomeServlet</servlet-name>
    <servlet-class>servlet.HomeServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HomeServlet</servlet-name>
    <url-pattern>/home</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>servlet.LogoutServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/logout</url-pattern>
  </servlet-mapping>
</web-app>
```

คำอธิบายโค้ด

- **LoginServlet**: ตรวจสอบข้อมูลผู้ใช้, ป้องกัน Session Fixation ด้วยการ invalidate session เก่า และสร้าง session ใหม่พร้อมเก็บ IP
- **HomeServlet**: ตรวจสอบ session ว่าถูกสร้างและ username อยู่ใน session หรือไม่, ตรวจสอบ IP Address เพื่อป้องกัน Session Hijacking
- **LogoutServlet**: ทำลาย session และกลับไปหน้า login
- **login.jsp**: หน้า login รับ username/password
- **home.jsp**: แสดงข้อความต้อนรับผู้ใช้
- **web.xml**: กำหนด servlet และ mapping

ผลการรัน

1. ผู้ใช้เข้าหน้า login.jsp
2. กรอกข้อมูลถูกต้อง (user/pass = user/pass)
3. ระบบสร้าง session ใหม่และเก็บ IP
4. เข้าสู่หน้า home.jsp แสดงข้อความต้อนรับ

5. ถ้า IP เปลี่ยน (จำลองเปลี่ยน IP) ระบบจะ logout อัตโนมัติ
6. ผู้ใช้สามารถ logout ได้ด้วยปุ่ม logout

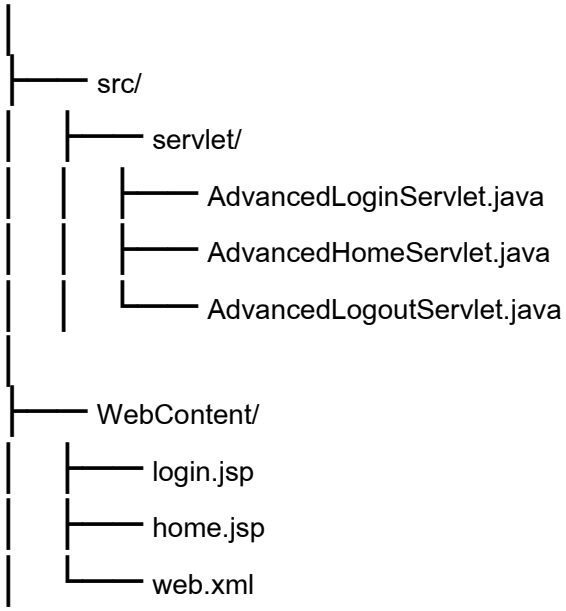
ตัวอย่างโปรแกรมแนวประยุกต์ 3 โปรแกรม

เสริมการป้องกันด้วย HttpOnly, Secure Cookie, และตรวจสอบ User-Agent

---

โครงสร้างโปรเจกต์แนวประยุกต์

SessionSecurityAdvanced/



---

1. AdvancedLoginServlet.java

```
package servlet;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class AdvancedLoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if ("user".equals(username) && "pass".equals(password)) {
```

```
            HttpSession oldSession = request.getSession(false);
            if (oldSession != null) oldSession.invalidate();

            HttpSession newSession = request.getSession(true);
            newSession.setAttribute("username", username);
            newSession.setMaxInactiveInterval(15 * 60);

            // เก็บ IP และ User-Agent สำหรับตรวจสอบ session hijacking
            newSession.setAttribute("clientIP", request.getRemoteAddr());
            newSession.setAttribute("userAgent", request.getHeader("User-Agent"));

            // ตั้งค่า JSESSIONID cookie HttpOnly และ Secure
            Cookie[] cookies = request.getCookies();
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    if ("JSESSIONID".equals(cookie.getName())) {
                        cookie.setHttpOnly(true);
                        cookie.setSecure(request.isSecure()); // ตั้ง Secure เฉพาะ HTTPS
                        response.addCookie(cookie);
                        break;
                    }
                }
            }

            response.sendRedirect("home");
        } else {
            request.setAttribute("errorMessage", "Invalid username or password");
            request.getRequestDispatcher("login.jsp").forward(request, response);
        }
    }
}
```

2. AdvancedHomeServlet.java

```
package servlet;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class AdvancedHomeServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        HttpSession session = request.getSession(false);

        if (session == null || session.getAttribute("username") == null) {
            response.sendRedirect("login.jsp");
            return;
        }

        String sessionIP = (String) session.getAttribute("clientIP");
        String sessionUA = (String) session.getAttribute("userAgent");
        String currentIP = request.getRemoteAddr();
        String currentUA = request.getHeader("User-Agent");

        if (!currentIP.equals(sessionIP) || !currentUA.equals(sessionUA)) {
            session.invalidate();
            response.sendRedirect("login.jsp");
            return;
        }

        request.getRequestDispatcher("home.jsp").forward(request, response);
    }
}
```

---

3. AdvancedLogoutServlet.java

```
package servlet;

import javax.servlet.*;
```

```
import javax.servlet.http.*;
import java.io.IOException;

public class AdvancedLogoutServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        HttpSession session = request.getSession(false);
        if (session != null) session.invalidate();
        response.sendRedirect("login.jsp");
    }
}
```

---

4. login.jsp และ home.jsp

เหมือนตัวอย่างพื้นฐาน

---

5. web.xml

เหมือนตัวอย่างพื้นฐาน แค่เปลี่ยนชื่อ servlet เป็น AdvancedLoginServlet, AdvancedHomeServlet, AdvancedLogoutServlet และ mapping url ให้เหมาะสม เช่น /advLogin, /advHome, /advLogout

---

คำอธิบายโค้ดแนวประยุกต์

- เพิ่มการตรวจสอบ **User-Agent** ร่วมกับ IP เพื่อเพิ่มความปลอดภัย
- ตั้งค่า Cookie HttpOnly และ Secure เพื่อป้องกันการขโมย Session Cookie ผ่าน XSS และให้ส่งผ่าน HTTPS เท่านั้น
- รักษา Session Timeout ไว้เช่นเดิม 15 นาที
- กระบวนการ login/logout เหมือนเดิม แต่ความปลอดภัยสูงขึ้น

ผลการรันแนวประยุกต์

- ระบบล็อกอินทำงานปกติ
- Session ถูกสร้างใหม่ทุกครั้งหลังล็อกอินเพื่อป้องกัน Session Fixation
- Cookie Session ตั้งค่า HttpOnly และ Secure (ถ้าใช้ HTTPS)
- หาก IP หรือ User-Agent เปลี่ยนแปลงระหว่างใช้งาน Session จะถูกยกเลิกและถูก Redirect กลับหน้า login
- ผู้ใช้สามารถ logout ได้ตามปกติ

การตั้งค่า Session Timeout และการจัดการ Session หมดอายุ

1. ความหมายของ Session Timeout

- **Session Timeout** คือระยะเวลาที่ Session จะยังคงใช้งานได้หลังจากไม่มีการใช้งาน (request) จากผู้ใช้
- เมื่อเวลานี้หมดลง **Session จะถูกยกเลิก (invalidate)** อัตโนมัติ
- ช่วยลดความเสี่ยงเรื่องความปลอดภัย เช่น การใช้งาน session ต่อจากผู้ใช้เดิมโดยไม่ได้รับอนุญาต

2. วิธีตั้งค่า Session Timeout

2.1 ตั้งค่าในไฟล์ web.xml

```
<session-config>
    <session-timeout>15</session-timeout>  <!-- เวลาเป็นนาที -->
</session-config>
```

- ตั้งเป็น 15 นาที หมายความว่า หากไม่มี request จาก client ภายใน 15 นาที session จะหมดอายุ

2.2 ตั้งค่า Timeout แบบโปรแกรม (ใน Servlet)

```
HttpSession session = request.getSession();
session.setMaxInactiveInterval(15 * 60);  // 15 นาที (วินาที)
```

- ตั้ง session timeout เฉพาะ session นั้น ๆ
- สามารถตั้งได้หลังจากสร้าง session แล้ว

3. การตรวจสอบ Session หมดอายุ

- เมื่อ Session หมดอายุแล้ว request.getSession(false) จะคืนค่าเป็น null
- ใน JSP/Servlet ควรตรวจสอบ session ก่อนใช้งานข้อมูล session

```
HttpSession session = request.getSession(false);
if (session == null || session.getAttribute("username") == null) {
    // session หมดอายุ หรือยังไม่ login
    response.sendRedirect("login.jsp");
    return;
}
```

4. การจัดการ Session หมดอายุอย่างเหมาะสม

- ควรแจ้งผู้ใช้เมื่อ session หมดอายุ เช่น Redirect ไปหน้า login พร้อมข้อความแจ้ง
- สามารถใช้ JavaScript ตรวจจับ session timeout ได้เช่นกัน (ผ่าน Ajax หรือ timer) เพื่อแจ้งผู้ใช้ล่วงหน้า
- หลีกเลี่ยงการใช้ session ที่หมดอายุเพราะจะเกิด NullPointerException

---

5. ตัวอย่างโค้ด JSP + Servlet

---

web.xml

```
<session-config>
    <session-timeout>10</session-timeout> <!-- หมดอายุหลัง 10 นาที -->
</session-config>
```

LoginServlet.java

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    if ("user".equals(username) && "pass".equals(password)) {
        HttpSession session = request.getSession(true);
        session.setAttribute("username", username);
        session.setMaxInactiveInterval(10 * 60);  // 10 นาที
        response.sendRedirect("home.jsp");
    } else {
        request.setAttribute("errorMessage", "Invalid credentials");
        request.getRequestDispatcher("login.jsp").forward(request, response);
    }
}
```

HomeServlet.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    HttpSession session = request.getSession(false);
```

```
    if (session == null || session.getAttribute("username") == null) {
        // Session หมดอายุหรือยังไม่ได้ login
        response.sendRedirect("login.jsp?timeout=true");
        return;
    }

    request.getRequestDispatcher("home.jsp").forward(request, response);
}
```

---

login.jsp (แจ้งเตือน Timeout)

```
<% String timeout = request.getParameter("timeout"); %>
<% if ("true".equals(timeout)) { %>
    <p style="color:red;">Your session has expired. Please login again.</p>
<% } %>
<form method="post" action="login">
    <!-- ฟอร์ม login -->
</form>
```

---

6. เทคนิคเสริม

- **Keep-Alive Ping:** ใช้ JavaScript ส่ง request ยืนยัน session ทุก ๆ ระยะเวลาหนึ่ง เพื่อป้องกัน session หมดอายุโดยไม่ได้ตั้งใจ (ถ้าต้องการ)
- **Automatic Logout:** แจ้งเตือนผู้ใช้ก่อน session หมดอายุจริง เช่น แจ้ง popup 1 นาทีล่วงหน้า
- **Session Listener:** ใช้ HttpSessionListener เพื่อติดตาม event การสร้างและทำลาย session เช่น บันทึก log การ logout อัตโนมัติเมื่อ session หมดอายุ

สรุป

การตั้งค่า	วิธีการ	ประโยชน์
Session Timeout (web.xml)	ตั้งค่า global session timeout ในไฟล์ web.xml	กำหนดเวลาหมดอายุ session โดยรวม
Session Timeout (Servlet)	ตั้งค่าเฉพาะ session ด้วย setMaxInactiveInterval()	ตั้งเวลาหมดอายุที่แตกต่างกันได้
ตรวจสอบ session	ใช้ getSession(false) เพื่อตรวจสอบ	ป้องกัน NullPointerException และการ

การตั้งค่า	วิธีการ	ประโยชน์
ก่อนใช้งาน	session อยู่หรือไม่	เข้าถึงข้อมูล session ที่หมดอายุ
แจ้งเตือนผู้ใช้	Redirect หรือแสดงข้อความเมื่อ session หมดอายุ	เพิ่มประสบการณ์ผู้ใช้ที่ดี

ต่อไปนี้คือตัวอย่างโปรแกรม JSP/Servlet **จำนวน 6 โปรแกรม (3 พื้นฐาน + 3 แนวประยุกต์)** ที่แสดงการ **ตั้งค่า Session Timeout และการจัดการ Session หมดอายุ** พร้อมโครงสร้างแบบเต็ม คำอธิบายโค้ด และผลการรัน:

☐ **หมวดพื้นฐาน (Basic Examples)**

☐ **โครงสร้างโปรเจกต์พื้นฐาน (ใช้ได้ทั้ง 3 ตัวอย่าง)**

```
SessionTimeoutBasic/
│
├── src/
│   └── servlet/
│       ├── LoginServlet.java
│       ├── HomeServlet.java
│       └── LogoutServlet.java
│
├── WebContent/
│   ├── login.jsp
│   ├── home.jsp
│   └── web.xml
```

☐ **โปรแกรมที่ 1: ตั้งค่า Timeout ใน web.xml**

☐ ***web.xml***

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">
  <session-config>
    <session-timeout>2</session-timeout> <!-- 2 นาที -->
  </session-config>
</web-app>
```

LoginServlet.java

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
    String user = req.getParameter("username");
    String pass = req.getParameter("password");
    if ("admin".equals(user) && "1234".equals(pass)) {
        HttpSession session = req.getSession(true);
        session.setAttribute("username", user);
        res.sendRedirect("home.jsp");
    } else {
        req.setAttribute("error", "Invalid credentials");
        req.getRequestDispatcher("login.jsp").forward(req, res);
    }
}
```

home.jsp

```
<%@ page session="true" %>
<%
String user = (String) session.getAttribute("username");
if (user == null) {
    response.sendRedirect("login.jsp?timeout=true");
}
%>
<h2>Welcome, <%= user %>!</h2>
<a href="logout">Logout</a>
```

login.jsp

```
<%
    String timeout = request.getParameter("timeout");
    if ("true".equals(timeout)) {
%>
    <p style="color:red;">Session expired. Please login again.</p>
<% } %>
<form action="login" method="post">
    Username: <input name="username" /><br>
    Password: <input name="password" type="password" /><br>
```

```
    <input type="submit" value="Login" />
</form>
```

☐ *ผลการรัน:*

- ล็อกอินสำเร็จ ⟶ เข้าสู่ home.jsp
- ถ้าไม่มี request ภายใน 2 นาที ⟶ กลับไปหน้า login พร้อมแจ้ง "Session expired"

☐ **โปรแกรมที่ 2: ตั้งค่า Timeout ด้วย setMaxInactiveInterval()**

```
HttpSession session = request.getSession(true);
session.setAttribute("username", user);
session.setMaxInactiveInterval(60); // 60 วินาที
```

☐ โค้ดอื่นเหมือนกับโปรแกรมที่ 1

☐ **โปรแกรมที่ 3: ตรวจสอบ session หมดอายุ + redirect อัตโนมัติ**

☐ *เพิ่มใน home.jsp (ตรวจสอบด้วย JavaScript)*

```
<script>
    // เตือนผู้ใช้เมื่อใกล้หมดเวลา
    setTimeout(function () {
        alert("Session is about to expire. Please save your work.");
    }, 50 * 1000); // 50 วินาที

    // Redirect อัตโนมัติเมื่อ session หมดอายุ
    setTimeout(function () {
        window.location.href = "login.jsp?timeout=true";
    }, 60 * 1000); // 60 วินาที
</script>
```

☐ **หมวดแนวประยุกต์ (Advanced Examples)**

☐ **โครงสร้างโปรเจกต์แนวประยุกต์**

```
SessionTimeoutAdvanced/
│
├── src/
│   └── controller/
│       ├── LoginController.java
│       ├── HomeController.java
```