

JSP - java server page

הסיבות לנחיצותו של יישום צד שרת בדפי JSP :

טיפול בנתונים באופן שיאפשר שמירה ושליפה שלהם

**שיקולי אבטחת מידע: שמירת הנתונים בצד שרת ממדרת את המידע ממשתמשים אחרים.
אם נשמור נתונים בדף **html**, הגולש יוכל לעיין בנתונים ע"י עיון בקוד בעזרת **view source**.**

ניהול וריכוז מידע על גולשים

טיפוסי משתנים

int n1 = 8;

- (1) הגדרת טיפוס משתנה מסוג מספר, שם המשתנה נקבע ע"י המפתח
- (2) בשונה מ-javascript חייבים להוסיף **int** לפני שם המשתנה
- (3) ניתן לבצע השמה של ערך מספרי ללא מירכאות
- (4) ניתן לבצע פעולות חישוביות עם משתנה זה

String s2 = "hello";

- (1) הגדרת טיפוס משתנה מסוג מחרוזת, שם המשתנה נקבע ע"י המפתח
- (2) בשונה מ-javascript חייבים להוסיף **String** לפני שם המשתנה
- (3) ניתן לבצע השמה רק עם ערך המוקף במירכאות
- (4) ניתן לבצע שרשור של משתנה זה עם מחרוזות אחרים

boolean b2 = true;

- (1) הגדרת טיפוס משתנה מסוג בוליאני, שם המשתנה נקבע ע"י המפתח
- (2) בשונה מ-javascript חייבים להוסיף **boolean** לפני שם המשתנה
- (3) ניתן לבצע השמה רק של ערכים של true/false
- (4) ניתן לבצע על משתנה זה פעולה בוליאנית (שקר,אמת)

טיפוסי פונקציות

```
public int calc(int a,int b)
{
```

```
    int sum = a+b;
```

```
    return sum;
```

```
}
public String printSum(int a,int b)
{
```

```
    int sum = a+b;
```

```
    return sum+"";
```

```
}
public boolean isBigger(int a,int b)
{
```

```
    if (a>b) return true;
```

```
    else return false;
```

```
}
public void calc(int a,int b)
{
```

```
    int sum = a+b;
```

```
    out.write("The result is "+sum);
```

```
}
```

טיפוס הפונקציה בהתאמה לטיפוס הערך המוחזר

טיפוס **void**, כלומר ללא ערך מוחזר

equals-מבצע השוואה בין אובייקטים

ב-javascript כאשר רצינו לבצע בדיקת השוואה בין שני משתנים

מסוג מחרוזות היינו מבצעים תנאי רגיל:

```
var s1 = "hello";  
var s2 = "thanks";  
if(s1 == s2) {  
    //do something  
}
```

ב-jsp על מנת לבצע בדיקת השוואה אנו משתמשים במתודה equal, לדוגמא:

```
String s1 = "hello";  
String s2 = "thanks";  
If(s1.equals(s2) == true) { //true אם שווה, מחזיר  
    //do something  
}
```

דוגמא של דף JSP - פעילות

חלק ההנחיות

// הגדרת שפת התיכנות

```
<%@page language="java"%>
```

// צורת קידוד בשרת

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```


המשך הדף: חלק הקוד – code section - צד שרת

<%!

String **output**;

// פונקציה שאינה מחזירה ערך void:

public void **Counter**(int **s**){

for(int i=0;i<s;i++){

output+= i + "
";

}

}

%>

<%

int sum=5;

Counter(**sum**); // קריאה לפונקציה

%>

הגדרת משתנים או פונקציות:
החלק הזה אינו מתבצע ב-**refresh**
של הדף.
אי אפשר לבצע פעולות מחוץ
לפונקציה.

פעולות קוד, חישובים וקריאות לפונקציות

המשך הדף: צד client

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <meta http-equiv="Content-Type" content="text/html;  
      charset=UTF-8">
```

```
    <title>JSP Page</title>
```

```
  </head>
```

```
  <body>
```

```
    <h1><%=output%></h1>
```

```
  </body>
```

```
</html>
```

תגית מיוחדת המאפשרת קישור
נתונים ל-html

הצגת משתנה המוגדר ב-jsp באמצעות html

עיצוב והצגת דף באמצעות JSP דרך א

<%

```
String showTbl="<table border='2'> ";  
for(int i=0;i<5;i++){  
    showTbl+="<tr><td>row"+i+"</td></tr>";  
}  
showTbl+="</table>";
```

הכנת המחרוזת לפני הצגתה
html=ב

%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>JSP Page</title>

</head>

<body>

<h1>My Table</h1>

<%=showTbl%>

הצגת המשתנה

</body>

</html>

עיצוב והצגת דף באמצעות JSP

דרך ב

```
<body>
  <h1> My table </h1>
  <table border="2">
    <%
      for(int i=0;i<5;i++)
      {
        <tr style="background-color: lightgrey">
          <td>
            row<%=i%>
          </td>
        </tr>
      }
    <%>
  </table>
</body>
```


קוד ה-jsp שמכין את הדף html מתבצע בשרת ולאחר מכן דף html מוכן וישלח ל-client

קוד ה-jsp מסוגל לשמור את הקשר בין חלקי הקוד, גם כאשר מפריד ביניהם קוד client (javascript, html)

פעילות:

בנה תוכנית ע"פ הקוד בדף זה, עם תוספת של צבע רקע המתחלף לאורך התאים

My table



row0
row1
row2
row3
row4

האובייקט request

אובייקט הבנוי בצד שרת ומטרתו לייצג מידע הקשור לבקשת הלקוח.

לאובייקט זה יש מספר פעולות שדרכן ניתן לגשת לנתוני הבקשה

1. request.getRemoteAddr() : המחזירה מחרוזת עם כתובת ה-

ip של המחשב שממנו בוצעה הבקשה.

2. request.getRemotePort() : המחזירה מחרוזת עם מספר

ה-port של הדפדפן שדרכו בוצעה הבקשה.

3. request.getParameter() : מחלצת מידע שנשלח לשרת

מטופס (form) באמצעות שיטת Get או שיטת Post

מתודת `getParameter` - פעילות

של אובייקט `request`

--first.html--

```
<form name="form1" action="second.jsp" onsubmit="return validation()"
method="post" >
Password:<input type="password" name="userPass" /> <br>
<input type="submit" value="send"/>
</form>
```

--second.jsp--

```
<%
    if(request.getParameter("userPass")!=null) {
        String pass = request.getParameter("userPass");
    }
%>
```

בדיקה אם נשלח מידע עם
name זה, אם לא, מחזיר **null**

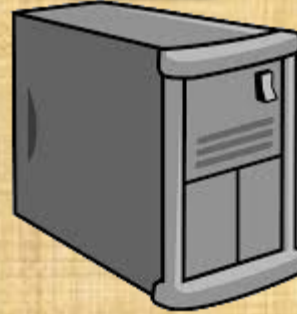


יתרונות של שיטת post

1. כמות המידע שאפשר להעביר, רבה יותר משיטת get
2. אפשר להעביר מידע רגיש מכיוון מכוון שבשיטה זו ניתן להסתיר את המידע מהגולשים, בניגוד לשיטת get שהמידע מופיע ב-URL

אם כן מתי נשתמש בשיטת get ?

שרת



השרת מחזיר לו
דף **html** עם
תגיות לפי הבקשה

השרת מעבד את
הנתונים ושולח
תגי **html** תאימים

מצב **stateless**
השרת מנתק מגע
עם הגולש

LoopBack

הגולש מבצע
בקשת
http get

לקוח



שליחת נתוני
ה-**form** ב-
submit

שיטת get

תג לתחילת QueryString

תג להוספת
parameter

--index.html - -

` 4+7 `

localhost:8080/WebsiteEx44_calculator/result.jsp?x=4&y=7

--result.jsp - -

`<%`

`String stX = request.getParameter("x");`

`String stY = request.getParameter("y");`

`int getX = Integer.parseInt(stX);`

`int getY = Integer.parseInt(stY);`

`int sum = getX+getY;`

`%>`

`html:`

`<body> The result is <%=sum%> </body>`

QueryString

לסיכום:

אם כך אנו רואים שאפשר לבנות

QueryString גם ע"י

לינק ולא רק ע"י submit

ולכן יש צורך בשיטת get

שיטת POSTBACK:

היא בקשה שמתרחשת לאותו דף בשרת,

למדנו שיטה להעברת נתונים ב-form לדף אחר בשרת בעזרת

. Submit

לעיתים נהיה מעוניינים לטפל בבקשות החוזרות ונשנות באותו דף

עצמו, ואז נשתמש בשיטת POSTBACK.

```
jsp:<%
```

```
String msg="Choose an action";  
if(request.getParameter("btnAdd")!=null){  
    msg = "Apparently you chose the Add operation ...";  
}  
if(request.getParameter("btnRemove")!=null){  
    msg = "Are you sure you want to delete?";  
}  
if(request.getParameter("btnUpdate")!=null){  
    msg = "Are you sure you want to update?";  
}
```

```
%>
```

```
html : <body>
```

```
<form name="form1" method="post" action="">
```

```
<%=msg%>
```

```
<input type="submit" name="btnAdd" value="Add" />
```

```
<input type="submit" name="btnRemove" value="Remove" />
```

```
<input type="submit" name="btnUpdate" value="Update" />
```

```
</form>
```

```
</body>
```

בדיקה איזה כפתור
נלחץ

הערך של ה-action ריק, כלומר
מתבצעת בקשת http post לדף
עצמו

פעילות:
בנה תוכנית ע"פ
הקוד בדף זה

מתודה נוספת של אובייקט request

request.getMethod() : בודק באיזה שיטה נשלחו נתוני ה-submit

לדוגמא:

```
if( request.getMethod().equals("GET") == true){
```

```
    //GET הנתונים נשלחו בשיטת
```

```
}
```

אובייקט response

בשונה מאובייקט **request** שמטרתו לייצג את המידע הקשור בבקשת הלקוח ,

אובייקט ה-**response** מיישם הוראות שגורמות לדפדפן לבצע פעולה כלשהיא.

1. `response.sendRedirect("target.jsp")` : גורם לדפדפן לבקש

מהשרת להעבירו לדף מסוים.

תוך כמה זמן לבקש מהשרת לעדכן\לשנות דף

2. `response.setHeader("Refresh","10;url='target.jsp'")` :

גורם לדפדפן לבצע http GET ל-`target.jsp` לאחר 10 שניות .

תכנות חוסר מצב (Stateless Programming)

כזכור , עם סיום טיפול של השרת בבקשת הלקוח, אז הלקוח והשרת מנתקים קשר כך שהשרת כלל אינו שומר את נתוני הלקוח.

שיטה זו מאפשרת טיפול במספר רב של בקשות במינימום עומס (היינו יכולים לתאר את העומס שהיה נגרם עקב שמירת כל פרטי גולש בשרת).

תכנות stateless מכילה מספר אובייקטים:

1. session : מתודה המאפשרת לשמור נתוני לקוח בין בקשות שונות לאותו דף או במעבר לדפים אחרים כל עוד ה- browser מופעל.
2. cookie : על מנת לשמור מידע בזיכרון לקוח-דפדפן.
3. application : מידע בזיכרון השרת כך שמידע זה יהיה נגיש לכל הלקוחות ולכל סוגי הבקשות.

אובייקט session דוגמא - פעילות

--index.jsp

JSP:

<%

```
if ( request.getParameter("User") != null) {  
    String userName = request.getParameter("User");  
    session.setMaxInactiveInterval(60*60);  
    session.setAttribute("user_name", userName);  
    response.sendRedirect("second.jsp");  
}
```

%>

HTML:

```
<body> My session is <%=session.getId()%><br>  
<form name="form1" action="index.jsp" method="post">  
User Name:<input type="text" name="User" /> <br>  
<input type="submit" value="send"/>  
</form>  
</body>
```

ה-**default** הוא
שה- **session** מתבטל
אם עבר פרק זמן של
30 דקות, מהבקשה
האחרונה (ללא
פעילות).
יש אפשרות להגדיר
את הזמן (בשניות),
אם מוגדר 1- כלומר
עד שסוגרים את
הדפדפן

השמת ערך לתוך **key** ב-**session**

המספר המזהה של
ה-**session**
המאותחל בכל
מעין **browser**

המשך דוגמא ליישום של session - פעילות

--second.jsp

JSP:

<%

String **userName**="";

if(**session**.isNew()==true){

session.invalidate();

response.sendRedirect("index.jsp");

}else{

userName = (String)**session**.getAttribute("user_name");

}

%>

HTML:

<body> My session is <%=**session**.getId()%>

<h1>Hello <%=**userName**%></h1>

</body>

security : בודק אם נוצר **session** חדש,
אם כן זה אומר שהגולש לא עבר דרך ה-**log in**
ולכן מעברים אותו לשם.

שולפים את הערך שמאוחסן
ב-**session** ע"פ ה-**key**

אובייקט application

בשונה מ-session id שה-id שלו והמידע שמאוחסן בו ייחודיים לדפדפן לקוח, מופע ה-application ומאפשר גישה למידע יחיד מכל הבקשות, מדפים שונים ומדפדפנים שונים.

שימוש לדוגמא :

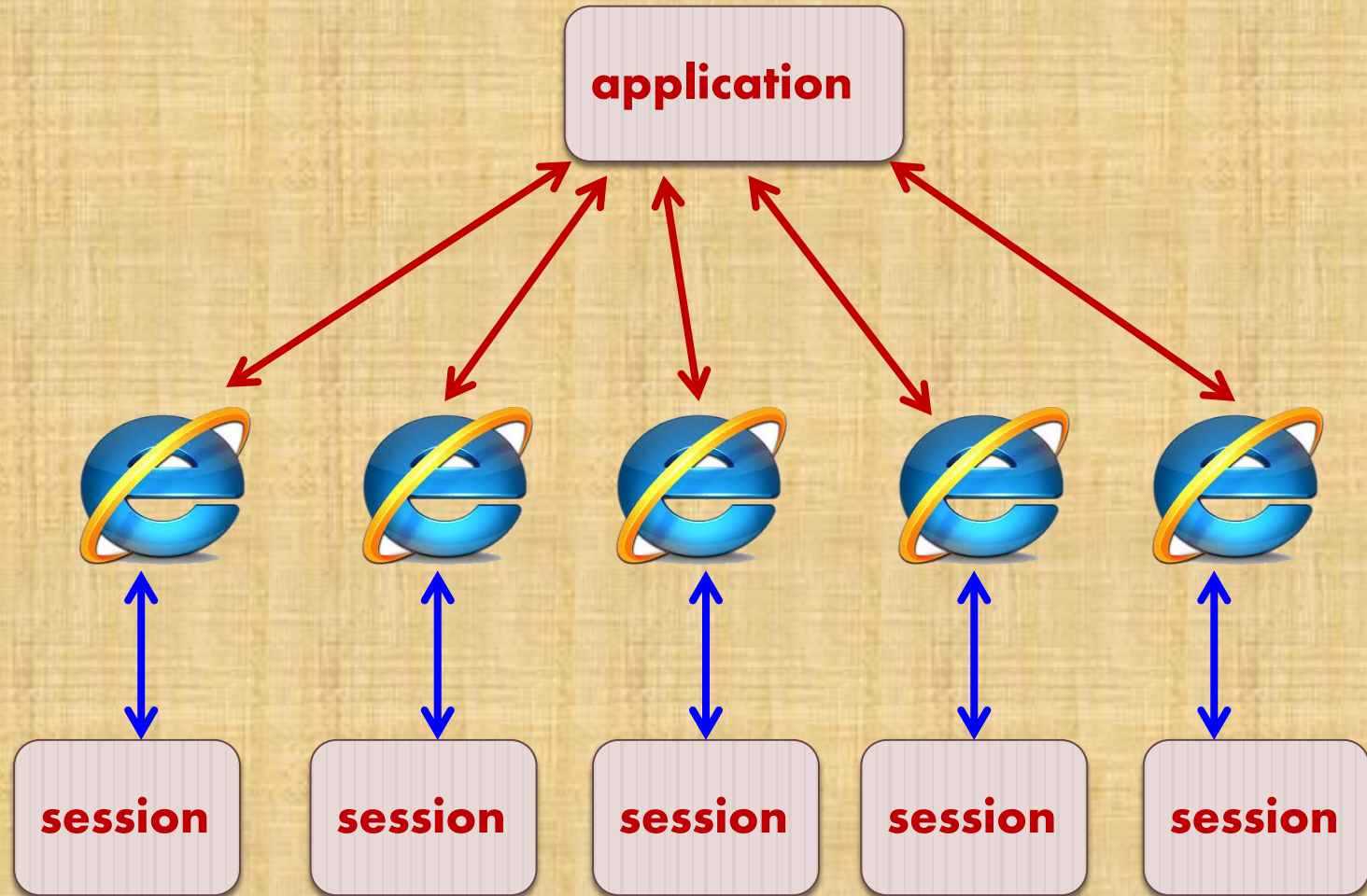
1) שמירת מידע בסגנון מפתח וערך:

```
application.setAttribute("Developer","Dorit");
```

2) גישה למידע:

```
String dev = (String)application.getAttribute("Developer");
```


יחס בין session, application, browser



אובייקט Cookie

בשונה מ-session ומ-application שנשמרים בשרת, ה-Cookie נשמר בזיכרון הדפדפן ולאחר מכן במחשב הלקוח. מאפשר לגשת למידע יחיד בבקשות שונות (ובדפדפנים שונים), לאותו אתר שהגדיר את ה-Cookie המסוים.

גודל Cookie יכול להגיע ל-4k וכל אתר יכול לשמור אצל הלקוח עד 20 Cookies.

הגדרת Cookie שליפתו

```
Cookie myCookie = new Cookie("keyName", "valueName");  
myCookie.setMaxAge(60 * 60 * 24 * 365);  
myCookie.setPath("/");  
response.addCookie(myCookie);
```

מגדיר שה-Cookie שיוכר בכל האתר

שומר Cookie ב-browser

```
Cookie[] arrCookie = request.getCookies();  
if(arrCookie!=null){  
    for (int i = 0; i < arrCookie.length; i++) {  
        if (arrCookie[i].getName().equals("keyName")) {  
            String varName = arrCookie[i].getValue();  
        }  
    }  
}
```

שולף את כל ה-Cookies שקיימים

מחלקת Random

מחלקה המאפשרת חישוב רנדומאלי של מספרים.

על מנת להשתמש במחלקה זו, יש צורך לייבא את ספריית util באופן הבא:

```
<%@page import = "java.util.*"%>
```

שימוש לדוגמא :

```
Random rnd = new Random();
```

```
int r1 = rnd.nextInt();
```

מחזיר ערכים בטווח טיפוס ה-**int**

```
int r1 = 1 + rnd.nextInt( 40 );
```

מחזיר ערכים בטווח **1-40**