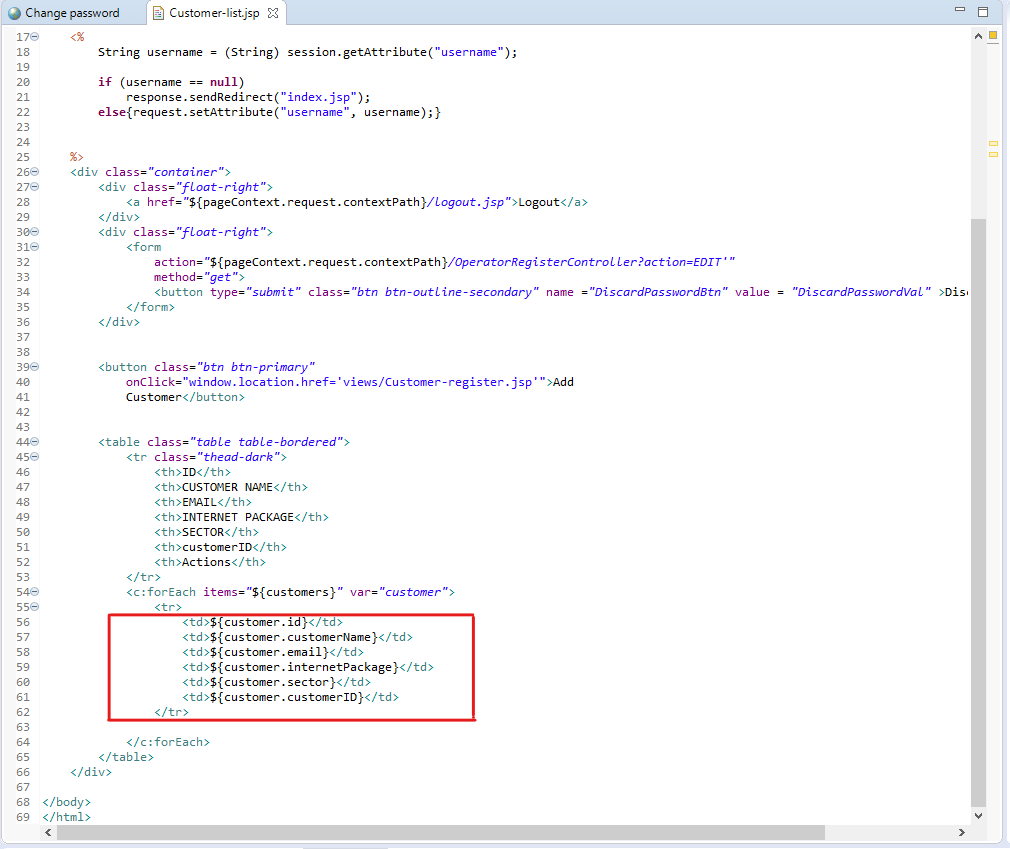
**חלק ב (שימוש בטכניקות XSS + SQL injection)**

**התקפת STORED XSS סעיף 4:**

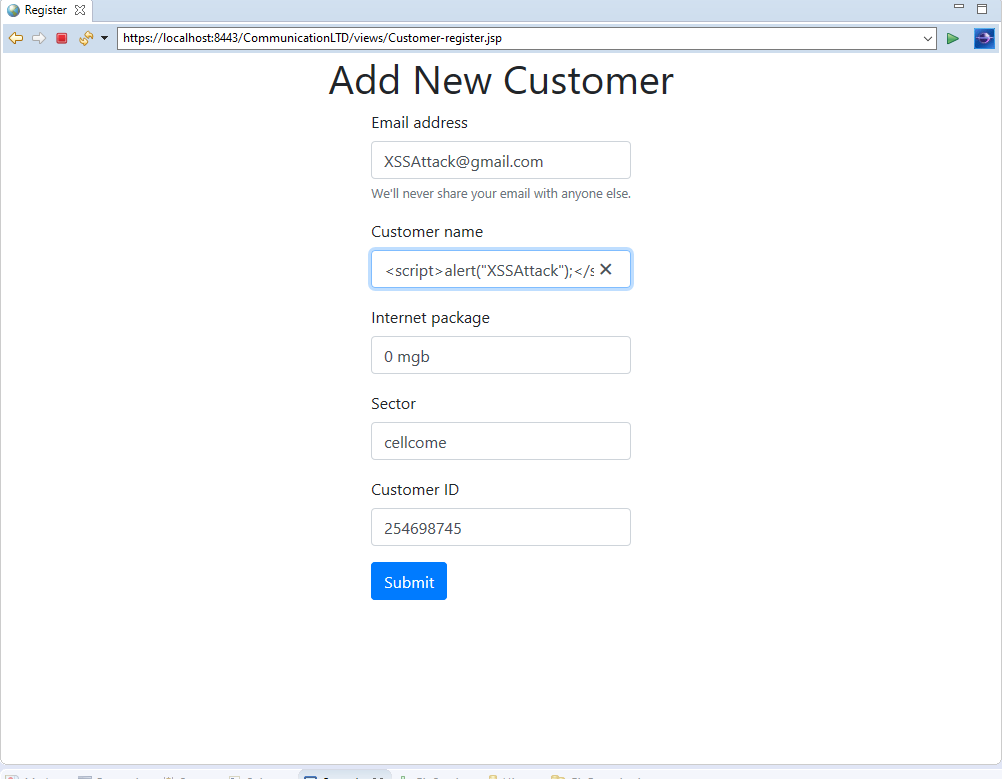
כידוע לנו, מסך המערכת, הכנסת לקוח חדש והצגתו עובד מול בסיס נתונים אשר נמצא ב mySql.  
אנו מזינים מידע על הלקוח ושמורים אותו בבסיס נתונים ולאחר מכן פונים לאותו הבסיס נתונים ומעדכים את הדף jsp בפרטי הלקוחות.  
דף ה jsp מחכה לקבל מידע על הלקוחות ורק לאחר מכן הוא מדפיס אותם.

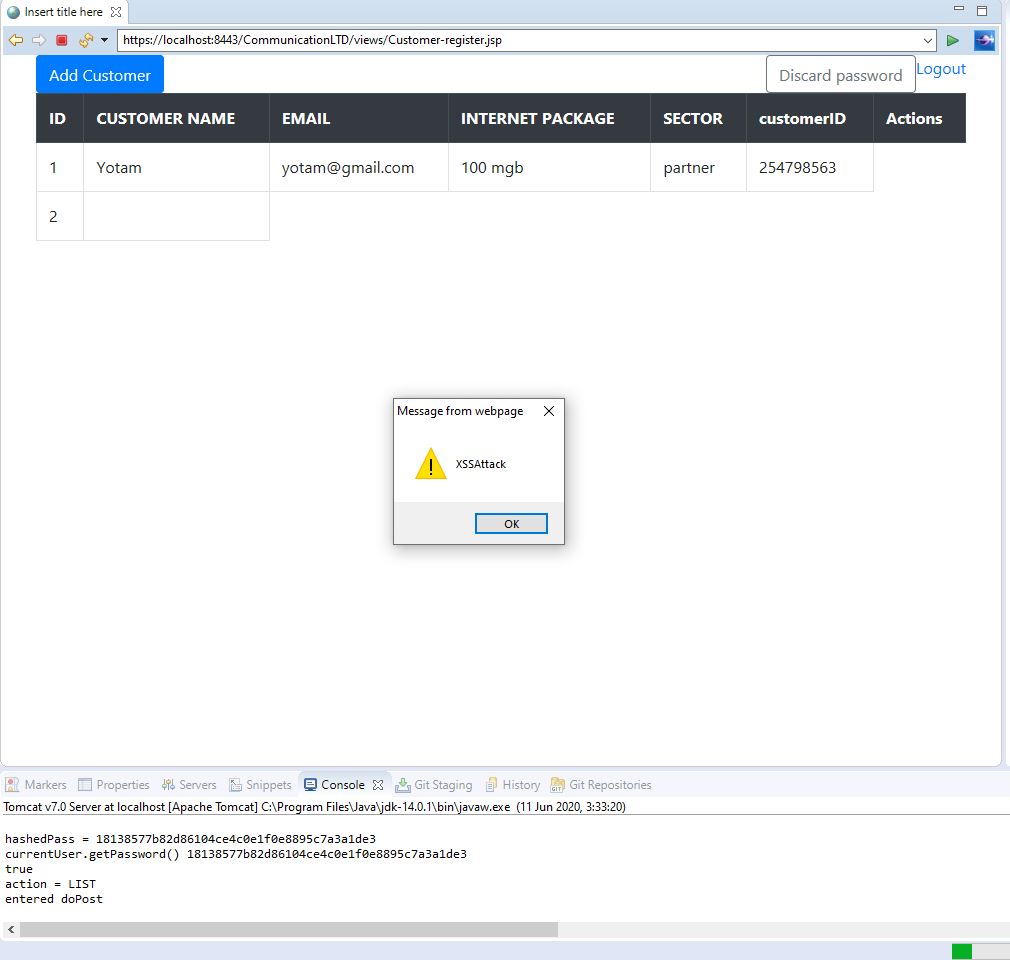
בדף Customer-list.jsp ניתן להבחין, באדום, שהדף מחכה למידע שישלח אליו (של הלקוח) כדי שיוכל להציג אותו על המסך.



נשתמש בחולשה זו וננסה להטמין script שלנו לתוך בסיס הנתונים, וברגע שמהערכת תפנה לבסיס הנתונים לקבלת מידע על הלקוחות בה, ה script שלנו יופעל!   
  
  
 ניצור לקוח חדש אך במקום שם, נכניס את ה script שלנו.

Customer name = <script>alert(“XSSAttack”);</script>



מיד לאחר הכנסת הלקוח לבסיס הנתונים, אנו חוזרים חזרה לדף רשימת הלקוחות.  
דף זה פונה לבסיס נתונים וממלא את הטבלה בלקוחות בעזרת ה JSP וברגע זה ה script שלנו מופעל!  
  
  


ניתן לראות שה script שלנו הופעל בהצלחה!

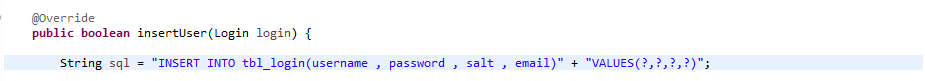
**התקפת SQL injection סעיף 1 + 3 + 4:**

**סעיף 1:**

בסעיף 1 מדבור על הכנסת משתמש חדש למערכת.   
כבר ברגע זה, אנו יכולים להניח שהכנסת המשתמש מתבצעת בשאילתה כגון:

INSERT INTO table\_name (col1,col2,col3 …. ) VALUES ( va1, val2, va3… );

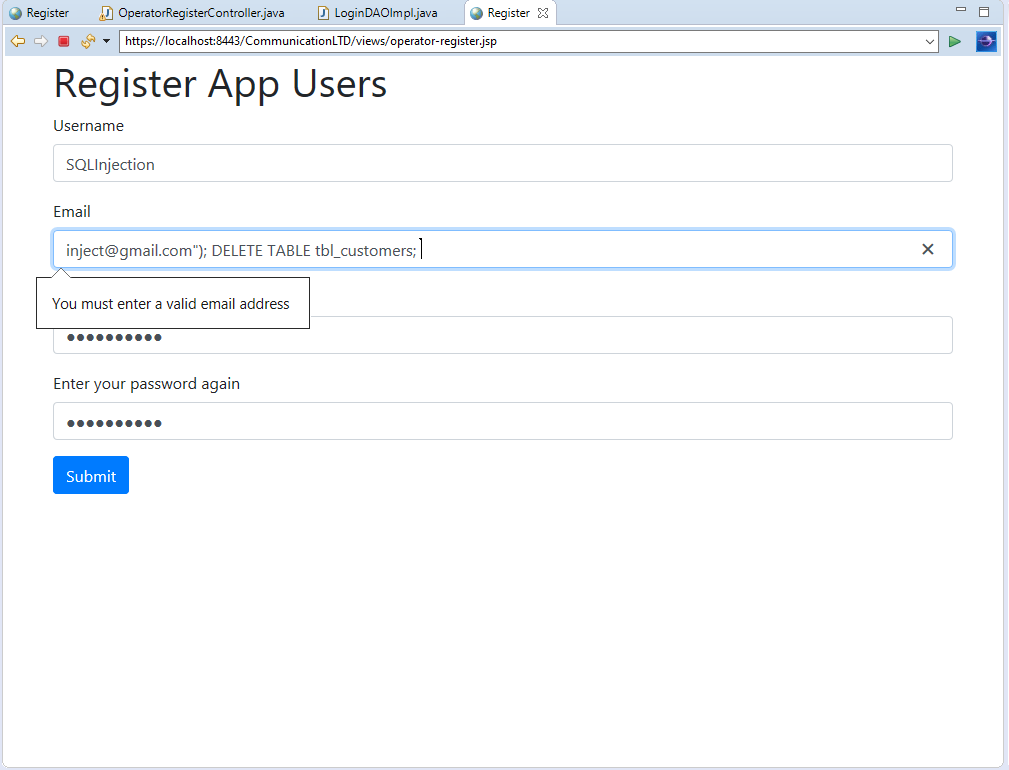
ואכן זוהי השאילתה:



ננסה ליצור שאילתת sql אשר תמחק את בסיס הנתונים.  
נניח וידוע לנו מה שמה של הטבלה שאנו רוצים לשנות.  
ננסה להכניס את הערך שלנו לערך האחרון ובוא לעשות את השינוי.   
השאיפה שלנו היא להגיע לשאילתה הבאה ולמחוק את טבלת הלקוחות:

INSERT INTO tbl\_login(username,password,salt,email)   
VALUES (“val1”,”val2”,”val3”,”**”); DELETE TABLE tbl\_customers;**

פעולת ההזרקה נכשלת!   
האתר אותו יצרנו, בודק את הקלט email, שהוא האחרון בהכנסה אל השאילת.   
לפני שהמערכת פונה את בסיס הנתונים ומנסה להרכיב את השאילת, היא בודקת האם הקלט הוא תקין.   
דבר זה מונע את ההתקפה!   
ללא בדיקה זו, התוקף היה מצליח למחוק את בסיס הנתונים של הלקוחות בהצלחה!

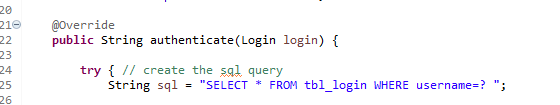


**סעיף 3:**

בסעיף 3 מדובר על Login, כלומר שניתן להניח שהשאילת תהיה מסוג

SELECT username,password FROM Login\_table WHERE username = \*\*\* and password = \*\*\*;

ואכן ניתן לראות דבר דומה:



\*חשוב לציין -   
אם השאילת אכן היתה נראת כך:

SELECT username,password FROM Login\_table WHERE username = \*\*\* and password = \*\*\*;

היינו יכולים בקלות להוסיף :

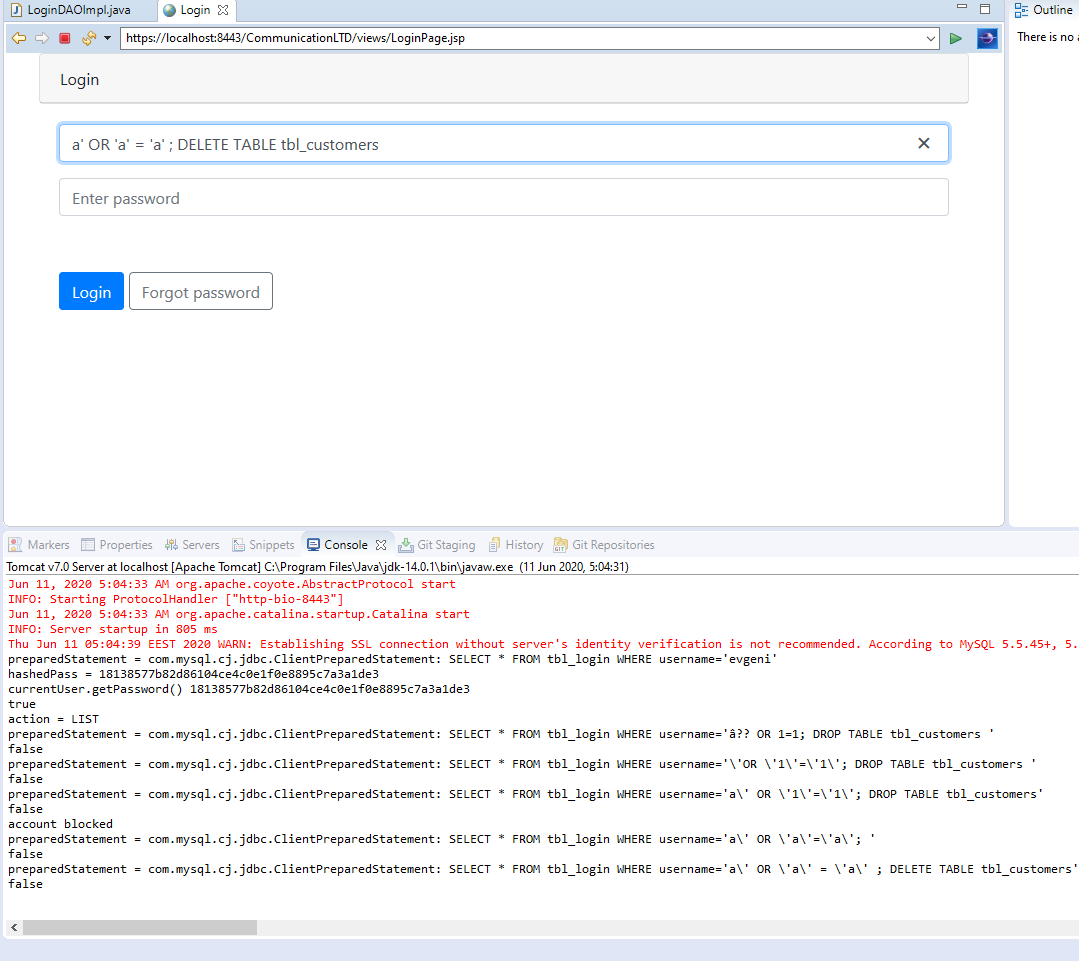
**“ OR 1 = 1;**

ולהכנס למערכת ללא סיסמא כלל!   
אך במערת שיצרנו הדבר אינו מתאפשר כיוון שהשאילת שלנו מחפשת האם היוזר קיים ומחזירה את פרטיו ולאכן מכן במחלקה אחרת מתבצעה הפיענוח של הסיסמא של המשתמש.

לכן במידה ודבר זה ידוע לנו מראש, ואנו יודעים את שמות הטבלאות נוכל לחבל בבסיס נתונים ולמחוק אותו או טבלאות ממנו, לדוגמא בעזרת:

**a‘ OR ‘1’=’1’; DROP TABLE tbl\_customers**

במערכת שלנו אנו משתמשים ב PreparedStatement.  
מחלקה זו מונעת מהקלט להיות חלק מהפקודה של SQL. כלומר שהקלט שהתוקף שולח עובר בדיקה ולא מאפשר לו הזרקה אל תוך השאילתה עצמה!



**סעיף 4:**

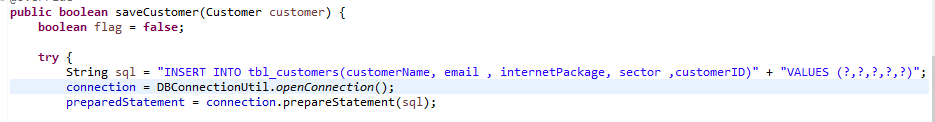
בסעיף 4 מדובר על הכנסת לקוחות אל הבסיס נתונים והצגתם, כלומר שאנו יכולים להניח שהשאילתות יהיו מהסוג:

INSERT INTO customer\_table (col1,col2,col3… ) VALUES (val1,val2,val3…);

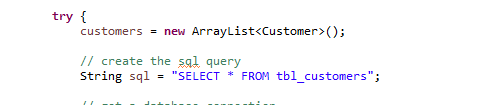
SELECT customer FROM customer\_table where CONDISION;

ואכן ניתן לראות:

שמירת לקוח:



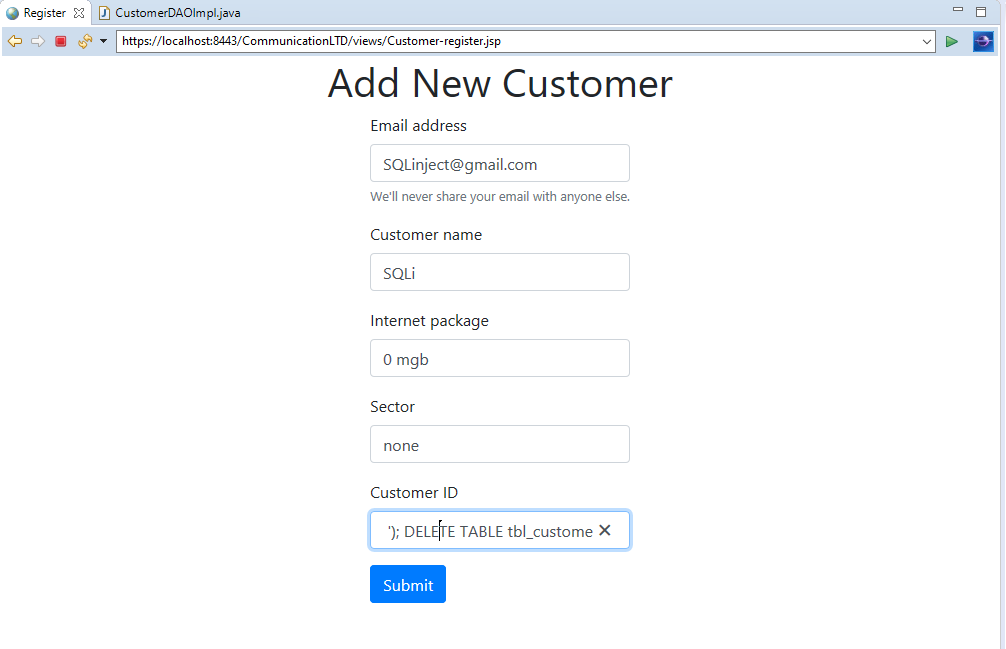
משיכת כל הלקוחות:

****

**נתבונן בשמירת לקוח.**  
במידה וידוע לנו שהערך האחרון הוא customerID, נוכל לבצע שם את ה SQL injection שלנו.

ננסה להזריק לתוך customerID ( הערך האחרון שמוכנס בשאילת ) את ה sql injection הבא:

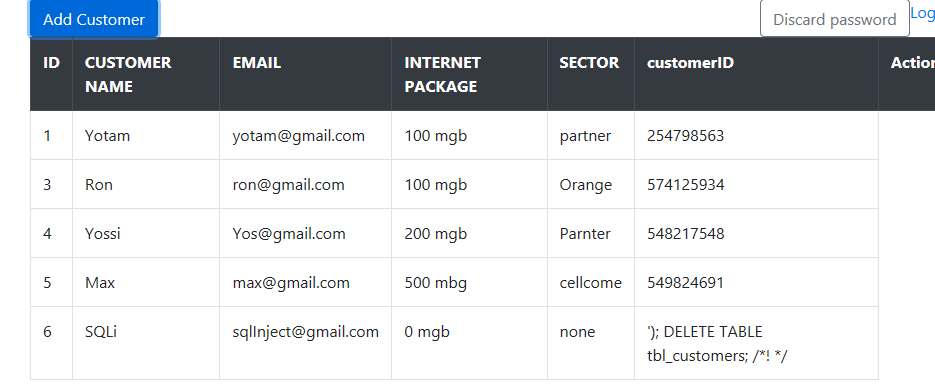
‘); DELETE TABLE tbl\_customer; /\*!\*/



ערך זה אמור להכנס אל תוך השאילת וליצור את השאילת הבאה:

INSERT INTO tbl\_login (customerName,email,internetPackage,sector,customerID) VALUES (‘customerName’,’email’,’internetPackage’,’sector’,’ **‘); DELETE TABLE tbl\_customers;** )

שאילת זו אמורה למחוק את כל בסיס הנתונים שלנו, אבל במערכת זו השתמשנו ב PreparedStatement אשר לא נותנת לקלט של המשתמש להכנס אל תוך השאילת של המערכת ובכך מנענו מחיקת טבלת לקוחות.





**נתבונן במשיכת לקוחות.**

השאילת מושכת את כל הנתונים שיש בטבלה ולא נותנת למשתמש גישה לשאילת או להחלטה על מה למשוך ולכן לא ניתן לתקוף את **משיכת לקוחות** בעזרת sql injection.

**הצגת פתרון נגד פרצות STORED XSS בסעיף 1 על ידי שימוש בקידוד תווים מיוחדים:**

במהלך בניית המערכת, אנו יודעים להגיד היכן דפי ה JSP אמורים לקבל מידע ולהפעיל אותו. אלו מקומות רגישים להתקפות XSS!

אחד הפתרונות לנקודות חולשה אלו הוא לבדוק בכל פעם את הערך שנשלח אל ה JSP לפני שליחתו ולבדוק האם המידע תקין.   
כלומר, בכל פעם, רגע לפני שנשלח מידע ונזמין דף JSP, אנו נעבוד על המידע שאמור להשלח ונחפש תווים מיוחדים כמון <>, </>.  
במידה ומצאנו תווים מיוחדים, נבדוק האם מדובר אכן בהתקפה ונחפש תווים ומחרוזות כגון script, documents, coockies וכו' ובכך נדע שאנו מותפים ולא נעביר מידע את דף ה JSP!

**הצגת פתרון נגד פרצות מסוג SQL injection בסעיפים 1 + 3 + 4 על ידי שימוש ב Parameter :**

התקפת ה SQL injection מחברת מחרוזת של התוקף אל כל הפקודה של השאילתה ובכך משנה את הערך המוחזר.   
אחד הפתרונות להתמודד עם בעיה זו היא להכניס את הקלט של המשתמש\תוקף לתוף Parameter.   
בכך, אנו לא נשרשר את הקלט של המשתמש\התוקף ישירות את תוך השאילת שלנו, אלא נוודא שהקלט שנשלח תקין, ונכניס אותו בתור פרמטר אל תוך השאילתה, ללא היכולת לשנות את השאילת עצמה.  
מעבר לכך, בעזרת Parameter המערכת עוברת קלט קלט בזמן ריצת השאילת, בודקת את תקינותו ורק אז מפעילה אותו.   
במערכת שלנו ניתן לראות שהשתמשנו ב PreparedStatement אשר הופך את הקלט של המשתמש ל Parameter ובכך אנו מונעים SQL injection.