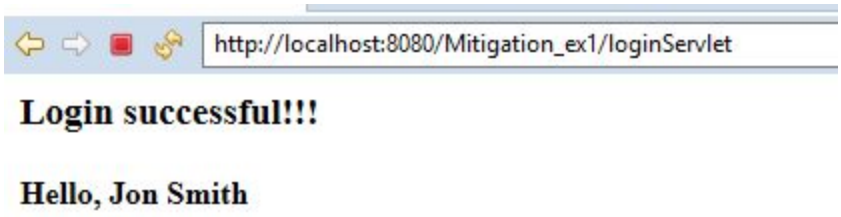
SQL Injection QUESTIONS:

Question 1: Is the website vulnerable to SQL injection attack? If yes, generate a SQL injection attack to login to the website without valid credentials. Attach a screenshot of the attack.

I logged in using SQL Injection. My attack payload was as follows.
SQL injection - any' or 1=1--



Question 2: Why the website is vulnerable to SQL injection attack?

It is taking the user input and placing it directly into the SQL query at the backend. If an attacker decides to manipulate the query as demonstrated above, then the SQL query at the backend will do something else than what it is expected to perform.

Question 3: Prevent SQL Injection vulnerability in the login form (use prepared statement). Attach a screenshot of the mitigation code.

```
public static boolean validate(String Name, String Pass) {
  boolean status = false;
  Connection conn = null;


   try {
        conn = getConnection();
       // Statement stmt = conn.createStatement();

      //  String query="select * from login where username='"+ Name+"' and password='"+Pass+"'";
        String query="select * from login where username=? and password=?";
        PreparedStatement stmt = conn.prepareStatement(query);
        stmt.setString(1, Name);
        stmt.setString(2, Pass);

        //ResultSet rs = stmt.executeQuery(query);
        ResultSet rs = stmt.executeQuery();
        status = rs.next();

      } catch (Exception e) {
        e.printStackTrace();
      } finally {
        if (conn != null) {
          try {
            conn.close();
          } catch (Exception e) {
          }
```

Question 4: Run the project again after mitigating the vulnerability and try to generate the same attack.
Attach a screenshot showing that this time login is unsuccessful.

XSS QUESTIONS:

Question 1: Is the website vulnerable to cross site scripting attack? If yes, what type of XSS exist the website? Generate the attack and attach a screenshot of the attack.

The Website is vulnerable to Reflected XSS attack in the username field of the Login page.
The payload was <script>alert(1);</script>

Question 2: Prevent XSS vulnerability. Attach a screenshot of the mitigation code. (Use both input validation and output encoding)

```
try {
    uName = ESAPI.validator().getValidInput("SanUn", request.getParameter("username"), "SafeString", 200, false);
    pWord = ESAPI.validator().getValidInput("SanUn", request.getParameter("userpass"), "SafeString", 200, false);
    System.out.println("Username by ESAPI : "+uName);

    if(LoginDao.validate(uName, pWord)){
        HttpSession session = request.getSession(false);
        if(session!=null){

        String fullName=LoginDao.getName(uName, pWord);

        session.setAttribute("name", fullName);
        }
        RequestDispatcher rd=request.getRequestDispatcher("welcome.jsp");
        rd.forward(request,response);


    }
    else{
        String safe = ESAPI.encoder().encodeForHTML("Username"+uName+" is not valid or "+ pWord+"is not valid");

        out.print("<p style=\"color:red\">"+safe+"</p>");
        RequestDispatcher rd=request.getRequestDispatcher("index.jsp");
        rd.include(request,response);
    }

    out.close();


} catch (ValidationException e) {
    // TODO Auto-generated catch block
    System.out.println("invalid input");
    //e.printStackTrace();
} catch (IntrusionException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Question 3: Run the project again after mitigating the vulnerability and try to generate the same attack.
Attach a screenshot showing that this time the attack is unsuccessful.