JavaEE Application

Day 5: Java EE Security (JDBC Realm) and JWT

Alan Biju
@itsmeambro

1. Java EE Security with Database Authentication (JDBC Realm)

Implementing authentication in Java EE using a database-backed security model ensures that users' credentials are stored securely and validated against a relational database like MySQL, PostgreSQL, or Oracle.

This guide covers:

✔ Form-Based Authentication using j_security_check
✔ JDBC Realm for Database-Based User Authentication
✔ Role-Based Access Control (RBAC) with Servlets
✔ Password Hashing for Secure Storage

1. Setting Up Database Authentication in Java EE

To authenticate users using a database, we follow these steps:

1️⃣ Create a users table to store username & hashed password
2️⃣ Create a roles table to assign user roles (admin, user, etc.)
3️⃣ Configure JDBC Realm in the application server
4️⃣ Implement Form-Based Authentication (j_security_check)
5️⃣ Secure Servlets using Role-Based Access Control

📌 Database Schema for Authentication

In a Java EE security model, user credentials are stored in a relational database with two tables:

🔹 users Table → Stores username & hashed password
🔹 user_roles Table → Maps users to specific roles
🔹 SQL Schema (MySQL Example)

```
CREATE TABLE users (
    username VARCHAR(50) PRIMARY KEY,
    password VARCHAR(255) NOT NULL  -- Store hashed passwords!
);
```


Alan Biju
@itsmeambro

```
CREATE TABLE user_roles (
    username VARCHAR(50),
    role_name VARCHAR(50),
    FOREIGN KEY (username) REFERENCES users(username)
);
```

📌 Configuring JDBC Realm in Tomcat

Java EE servers like Tomcat, WildFly, or GlassFish support JDBC Realm authentication, which validates credentials against a database.
To set up JDBC authentication in Apache Tomcat, modify *conf/server.xml*:

🔹 Configure server.xml

```
<Realm className="org.apache.catalina.realm.JDBCRealm"
       driverName="com.mysql.cj.jdbc.Driver"
       connectionURL="jdbc:mysql://localhost:3306/mydb"
       connectionName="root"
       connectionPassword="password"
       userTable="users"
       userNameCol="username"
       userCredCol="password"
       userRoleTable="user_roles"
       roleNameCol="role_name"
       digest="SHA-256" />
```

✔ Maps database users & roles to Java EE security
✔ Uses SHA-256 hashing for password comparison
✔ Supports multiple user roles for access control

📌 Form-Based Authentication with j_security_check

In Java EE, form-based authentication enables users to log in via a web form.The login form posts credentials to j_security_check, which Java EE handles.

🔹 Login Form (login.html)

```
<form method="POST" action="j_security_check">
    <label>Username:</label
```


Alan Biju
@itsmeambro

```
    <input type="text" name="j_username" required>
    <label>Password:</label>
    <input type="password" name="j_password" required>
    <button type="submit">Login</button>
</form>
```

✔ Uses POST request to j_security_check

✔ No need to write explicit authentication logic

📌 Securing Web Pages & Servlets

🔹 Securing URLs with web.xml

We define authentication & authorization rules in web.xml.

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>JDBCRealm</realm-name>
    <form-login-config>
        <form-login-page>/login.html</form-login-page>
        <form-error-page>/error.html</form-error-page>
    </form-login-config>
</login-config>
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Secure Area</web-resource-name>
        <url-pattern>/secure/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>

<security-role>
    <role-name>admin</role-name>
</security-role>
```

✔ Only users with role admin can access /secure/*

✔ Redirects users to login page if unauthenticated

✔ After login, users are redirected to protected resources

Alan Biju

@itsmeambro

🔹 Securing Servlets with Role-Based Access Control

Java EE allows role-based security for servlets using @RolesAllowed.

```
@WebServlet("/admin")
@DeclareRoles({"admin", "user"})
public class AdminServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws IOException {
        response.getWriter().println("Welcome, Admin! You have secure
access.");
    }
}
```

✔ Only "admin" role users can access /admin

✔ Java EE automatically validates user roles

📌 Storing Hashed Passwords in Database

Storing plain-text passwords is a huge security risk!

Use BCrypt hashing for secure password storage.

🔹 Hashing Passwords in Java

```
import org.mindrot.jbcrypt.BCrypt;

public class PasswordUtil {
    public static String hashPassword(String password) {
        return BCrypt.hashpw(password, BCrypt.gensalt());
    }

    public static boolean verifyPassword(String password, String
hashedPassword) {
        return BCrypt.checkpw(password, hashedPassword);
    }
}
```


Alan Biju
@itsmeambro

✔ Store hashPassword("mypassword") in the database

✔ Verify user login using:

```
boolean isValid = verifyPassword(inputPassword, dbHashedPassword);
```

📌 **Implementing Logout in Java EE**

To log out a user, invalidate the session.

🔹 **Logout Servlet**

```
@WebServlet("/logout")
public class LogoutServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws IOException {
        request.getSession().invalidate();
        response.sendRedirect("login.html");
    }
}
```

✔ Destroys the session

✔ Redirects user to login page

Alan Biju

@itsmeambro

2. JWT Token Authentication in Java EE

JWT (JSON Web Token) is widely used for stateless authentication in Java EE applications. Here's how you can implement JWT authentication in a Java EE app.

🔹 **Steps to Implement JWT in Java EE**

1️⃣ Generate JWT Token on Login
2️⃣ Validate JWT Token in Secured Endpoints
3️⃣ Use a JWT Filter for Authentication
4️⃣ Secure API Endpoints with JWT

📌 Add Dependency

If using Maven, add the following dependency:

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.11.5</version>
</dependency>
```

📌 Generate JWT Token (Login API)

This method generates a JWT token when the user logs in.

```
public class JwtUtil {
    private static final String SECRET_KEY = "###Key###";
    private static final long EXPIRATION_TIME = 86400000;

    public static String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username) // Set username as subject
                .setIssuedAt(new Date()) // Set issued date
                .setExpiration(new Date(System.currentTimeMillis()
                     + EXPIRATION_TIME))
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }
```


Alan Biju
@itsmeambro

```
    public static String validateToken(String token) {
        try {
            return Jwts.parser()
                    .setSigningKey(SECRET_KEY)
                    .parseClaimsJws(token)
                    .getBody()
                    .getSubject();
        } catch (Exception e) {
            return null; // Invalid token
        }
    }
}
```

📌 Login API (Returns JWT Token)

his servlet validates user credentials and returns a JWT token.

```
@WebServlet("/login")
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");

        // Dummy validation (Replace with database check)
        if ("admin".equals(username) && "password".equals(password)) {
            String token = JwtUtil.generateToken(username);

            response.setContentType("application/json");
            response.getWriter().write("{\"token\": \"" + token + "\"}");
        } else {
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            response.getWriter().write("Invalid credentials");
        }
    }
}
```


Alan Biju
@itsmeambro

📌 Protect API Endpoints Using JWT

```
@WebFilter("/secure/*")
public class JwtFilter implements Filter {
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {

        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        String authHeader = req.getHeader("Authorization");

        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            res.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            res.getWriter().write("Missing or invalid Authorization
                  header");
            return;
        }

        String token = authHeader.substring(7); // Remove "Bearer " prefix
        String username = JwtUtil.validateToken(token);

        if (username == null) {
            res.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            res.getWriter().write("Invalid or expired token");
            return;
        }

        request.setAttribute("username", username);
        chain.doFilter(request, response);
    }
}
```


Alan Biju
@itsmeambro

If you find this helpful, like and share it with your friends

Alan Biju
@itsmeambro