Uygulama Güvenliği İçin Aldığım Önlemler:

Java projelerinde güvenliği sağlamak için aşağıdaki önlemleri alıyorum:

Girdi Doğrulama ve Güvenli Kodlama

- Kullanıcılardan gelen verileri asla güvenilir kabul etmiyorum. Input validation (girdi doğrulama) için:
  - Whitelist (Beyaz Listeleme) yöntemini kullanıyorum.
  - Regular Expressions (Regex) ile girdileri filtreliyorum.
  - OWASP Java Encoder Library veya Apache Commons Validator gibi kütüphaneleri kullanarak güvenli girdileri doğruluyorum.

Cross-Site Scripting (XSS) Önlemleri

- **XSS saldırılarını önlemek için:**
  - HTML, JavaScript veya CSS içine gömülen kullanıcı girdilerini escape ediyorum (örneğin, &, <, > karakterlerini encode ediyorum).
  - Content Security Policy (CSP) kullanarak güvenli içerik kaynaklarını belirtiyorum.
  - OWASP'ın Java HTML Sanitizer gibi güvenlik kütüphanelerini kullanıyorum.

Örnek Güvensiz Kod (XSS Açığı Bulunan Java Kod Parçası):

```
String userInput = request.getParameter("username");
out.println("<h1>Hoşgeldiniz, " + userInput + "</h1>");
```

Güvenli Versiyon:

```
import org.owasp.encoder.Encode;
String userInput = request.getParameter("username");
out.println("<h1>Hoşgeldiniz, " + Encode.forHtml(userInput) + "</h1>");
```

- **SQL Injection Önlemleri**

Hazırlıklı (Parameterized) SQL Sorguları Kullanıyorum:

  - JDBC'de PreparedStatement ile parametreleri dışardan ayırarak SQL injection saldırılarını önlüyorum.
  - Hibernate gibi ORM araçları kullanıyorsam HQL (Hibernate Query Language) yerine Criteria API tercih ediyorum.

Örnek Güvensiz Kod (SQL Injection Açığı Bulunan Java Kod Parçası):

```
String query = "SELECT * FROM users WHERE username = '" + username + "' AND password = '" + password + "'";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

Güvenli Versiyon:

```
String query = "SELECT * FROM users WHERE username = ? AND password = ?";
PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString(1, username);
pstmt.setString(2, password);
ResultSet rs = pstmt.executeQuery();
```

Yetkilendirme ve Kimlik Doğrulama Önlemleri

- JWT (JSON Web Token) veya OAuth 2.0 ile güvenli kimlik doğrulama sağlıyorum.
- Spring Security gibi kütüphanelerle kullanıcı yetkilendirmesini (RBAC - Role-Based Access Control) yönetiyorum.
- Güçlü hash algoritmaları (örn. SHA256) ile şifreleri güvenli saklıyorum.

Java projelerimde uygulama güvenliğini sağlamak için en önemli güvenlik önlemlerini alıyorum. Kullanıcı girdilerini doğruluyor, XSS ve SQL Injection gibi saldırılara karşı güvenli kod yazıyorum. Örneğin, SQL Injection'a karşı PreparedStatement, XSS'e karşı OWASP Encoder kullanıyorum. Yetkilendirme için Spring Security ve OAuth 2.0 gibi çözümler uyguluyorum. Ayrıca, güvenlik testleri yaparak açıkları düzenli olarak kontrol ediyorum. Böylece uygulamanın güvenliğini maksimum seviyeye çıkarmış oluyorum.