Để đảm bảo tính bảo mật và toàn vẹn dữ liệu cho Hệ thống Quản lý Nhân sự, nhóm đã triển khai và áp dụng chặt chẽ các kỹ thuật lập trình phòng ngừa sau:

**a) Prepared Statements (Đối phó với SQL Injection)**

* **SQL Injection là gì?** SQL Injection (tấn công tiêm nhiễm SQL) là một trong những lỗ hổng bảo mật phổ biến và nguy hiểm nhất trên các ứng dụng web. Nó xảy ra khi kẻ tấn công chèn các mã SQL độc hại vào các trường dữ liệu đầu vào của ứng dụng (ví dụ: username, password, trường tìm kiếm) mà không được xử lý đúng cách. Khi ứng dụng thực thi truy vấn SQL có chứa mã độc này, kẻ tấn công có thể thao túng cơ sở dữ liệu (đọc, sửa, xóa dữ liệu nhạy cảm, hoặc thậm chí chiếm quyền kiểm soát máy chủ).
* **Prepared Statements là gì?** Prepared Statements (hay còn gọi là câu lệnh chuẩn bị) là một tính năng được cung cấp bởi các hệ quản trị cơ sở dữ liệu và các ngôn ngữ lập trình (như PHP với MySQLi hoặc PDO) để thực hiện các truy vấn SQL một cách an toàn và hiệu quả.
* **Cách thức hoạt động và tại sao lại thiết yếu:**
  1. **Tách biệt mã SQL và dữ liệu:** Thay vì ghép nối trực tiếp dữ liệu người dùng vào chuỗi truy vấn SQL, Prepared Statements hoạt động bằng cách gửi mẫu truy vấn (query template) đến cơ sở dữ liệu trước, trong đó các vị trí dữ liệu được đánh dấu bằng các tham số giữ chỗ (placeholders, thường là ?).
  2. **Biên dịch trước:** Cơ sở dữ liệu biên dịch mẫu truy vấn này và hiểu rõ cấu trúc của nó mà không cần biết giá trị thực của dữ liệu.
  3. **Gửi dữ liệu riêng biệt:** Sau đó, dữ liệu người dùng được gửi riêng biệt và được liên kết (bind) an toàn vào các tham số giữ chỗ. Cơ sở dữ liệu coi dữ liệu này là các giá trị thuần túy, chứ không phải là một phần của mã SQL.
  4. **Ngăn chặn thực thi mã độc:** Do dữ liệu và mã SQL được xử lý riêng biệt, bất kỳ ký tự đặc biệt nào trong dữ liệu người dùng (ví dụ: dấu nháy đơn ', dấu chấm phẩy ; được dùng để ngắt lệnh SQL) cũng sẽ được coi là một phần của giá trị chuỗi, chứ không phải là lệnh SQL, từ đó vô hiệu hóa mọi nỗ lực tiêm nhiễm mã độc.

Trong hệ thống của chúng tôi, **tất cả các thao tác tương tác với cơ sở dữ liệu có chứa dữ liệu đầu vào từ người dùng (ví dụ: đăng ký, đăng nhập, cập nhật thông tin, thêm/xóa nhân viên)** đều sử dụng Prepared Statements để đảm bảo an toàn tuyệt đối trước các cuộc tấn công SQL Injection.

**b) Băm mật khẩu (Password Hashing) (Đối phó với Lộ lọt mật khẩu)**

* **Vấn đề:** Lưu trữ mật khẩu dưới dạng văn bản thuần túy (plain text) trong cơ sở dữ liệu là cực kỳ nguy hiểm. Nếu CSDL bị tấn công và lộ ra, tất cả mật khẩu người dùng sẽ bị phơi bày, gây ra hậu quả nghiêm trọng về bảo mật thông tin cá nhân.
* **Băm mật khẩu là gì?** Băm mật khẩu là quá trình chuyển đổi một chuỗi văn bản (mật khẩu) thành một chuỗi ký tự có độ dài cố định (hash) bằng cách sử dụng một thuật toán băm một chiều (one-way cryptographic hash function).
  + **Một chiều:** Không thể đảo ngược từ chuỗi hash để lấy lại mật khẩu gốc.
  + **Tạo muối (Salting):** Các hàm băm hiện đại (như password\_hash() trong PHP) tự động thêm một "muối" (salt) ngẫu nhiên vào mật khẩu trước khi băm. Muối này đảm bảo rằng hai người dùng có cùng mật khẩu sẽ tạo ra hai chuỗi hash hoàn toàn khác nhau, chống lại các cuộc tấn công bảng cầu vồng (rainbow table attacks).
  + **Chi phí tính toán (Cost factor/Work factor):** Các hàm băm cũng cho phép cấu hình một yếu tố chi phí (cost factor) để làm cho quá trình băm tốn thời gian hơn. Điều này làm chậm đáng kể các cuộc tấn công brute-force hoặc dictionary attack trên các chuỗi hash đã bị lộ.
* **Cách thức hoạt động và tại sao lại thiết yếu:**
  + **Khi đăng ký/cập nhật mật khẩu:** Mật khẩu do người dùng nhập vào sẽ được băm bằng hàm password\_hash() (ví dụ: password\_hash($password, PASSWORD\_BCRYPT)). Chỉ chuỗi hash này được lưu vào CSDL.
  + **Khi đăng nhập:** Mật khẩu người dùng nhập vào sẽ được băm lại bằng cùng thuật toán và sau đó so sánh với chuỗi hash đã lưu trong CSDL bằng hàm password\_verify() (ví dụ: password\_verify($input\_password, $stored\_hash)). Hàm này sẽ trả về true nếu khớp, false nếu không.
  + **Thiết yếu vì:** Ngay cả khi cơ sở dữ liệu bị xâm nhập và kẻ tấn công có được tất cả các chuỗi hash mật khẩu, chúng cũng không thể biết được mật khẩu gốc. Điều này bảo vệ người dùng khỏi việc bị lộ thông tin cá nhân và bị tấn công bằng mật khẩu đó trên các dịch vụ khác.

**c) htmlspecialchars() (Đối phó với Cross-Site Scripting - XSS)**

* **XSS là gì?** Cross-Site Scripting (XSS) là một loại tấn công tiêm nhiễm mã kịch bản phía client (ví dụ: JavaScript) vào các trang web được người dùng khác xem. Kẻ tấn công có thể chèn các mã độc này thông qua các trường nhập liệu (như tên, mô tả, bình luận) nếu ứng dụng không xử lý đầu ra đúng cách. Khi người dùng khác truy cập trang web có chứa mã độc, trình duyệt của họ sẽ thực thi mã đó, cho phép kẻ tấn công đánh cắp cookie, session token, thay đổi nội dung trang, hoặc chuyển hướng người dùng đến các trang lừa đảo.
* **htmlspecialchars() là gì?** htmlspecialchars() là một hàm trong PHP được sử dụng để chuyển đổi các ký tự đặc biệt trong HTML thành các thực thể HTML (HTML entities).
  + & (ampersand) thành &amp;
  + " (double quote) thành &quot;
  + ' (single quote) thành &#039; hoặc &apos;
  + < (less than) thành &lt;
  + > (greater than) thành &gt;
* **Cách thức hoạt động và tại sao lại thiết yếu:** Khi hiển thị bất kỳ dữ liệu nào được lấy từ cơ sở dữ liệu hoặc từ dữ liệu đầu vào của người dùng ra giao diện HTML, việc truyền dữ liệu đó qua hàm htmlspecialchars() là bắt buộc. Hàm này đảm bảo rằng bất kỳ ký tự nào có thể được hiểu là một phần của mã HTML hoặc JavaScript sẽ được chuyển đổi thành dạng an toàn (chỉ hiển thị dưới dạng văn bản).
  + **Thiết yếu vì:** Nếu kẻ tấn công nhập <script>alert('XSS');</script> vào trường tên người dùng, và chuỗi này không được htmlspecialchars() xử lý khi hiển thị, trình duyệt sẽ coi đây là một đoạn mã JavaScript và thực thi nó. Ngược lại, nếu được xử lý, nó sẽ chỉ hiển thị thành &lt;script&gt;alert(&#039;XSS&#039;);&lt;/script&gt;, vô hiệu hóa cuộc tấn công.

Trong dự án, htmlspecialchars() được sử dụng ở nhiều nơi, đặc biệt là khi hiển thị tên người dùng, email, hoặc các thông tin khác trên các trang như employee.php, employee-list.php, update-information.php.

**d) Quản lý phiên (Session Management) (Đối phó với Truy cập trái phép & Phân quyền)**

* **Vấn đề:** Một ứng dụng web cần biết ai đang truy cập và vai trò của họ để kiểm soát quyền truy cập vào các chức năng và dữ liệu nhạy cảm. Nếu không có cơ chế quản lý phiên hiệu quả, kẻ tấn công có thể mạo danh người dùng hoặc truy cập vào các tài nguyên không được phép.
* **Session là gì?** Session (phiên làm việc) là một cơ chế giúp máy chủ lưu trữ thông tin về trạng thái tương tác của người dùng trong suốt nhiều yêu cầu HTTP. Mỗi người dùng được gán một ID phiên duy nhất, thường được lưu trữ trong một cookie trên trình duyệt của họ. Thông tin thực tế của session (ví dụ: ID người dùng, vai trò, trạng thái đăng nhập) được lưu trữ trên máy chủ, an toàn hơn.
* **Cách thức hoạt động và tại sao lại thiết yếu:**
  1. **Khởi tạo phiên (session\_start()):** Ở đầu mỗi trang PHP cần sử dụng session, hàm session\_start() được gọi để khởi động hoặc tiếp tục một phiên làm việc.
  2. **Lưu trữ thông tin phiên:** Sau khi người dùng đăng nhập thành công, các thông tin quan trọng như user\_id và role của họ được lưu vào biến siêu toàn cục $\_SESSION.
     + $\_SESSION['user\_id'] = $user\_id;
     + $\_SESSION['role'] = $role\_id; (ví dụ: 0 cho Manager, 1 cho Employee)
  3. **Kiểm tra phân quyền:** Trên mỗi trang yêu cầu phân quyền, hệ thống kiểm tra các giá trị trong $\_SESSION.
     + if (!isset($\_SESSION['user\_id'])): Kiểm tra xem người dùng đã đăng nhập chưa. Nếu chưa, chuyển hướng về trang đăng nhập.
     + if ($\_SESSION['role'] != 0): Kiểm tra vai trò của người dùng. Ví dụ, trang employee-list.php chỉ cho phép người có role = 0 (Manager) truy cập. Nếu không đúng vai trò, chuyển hướng đến trang không có quyền.
  4. **Kết thúc phiên (session\_destroy()):** Khi người dùng đăng xuất, session sẽ bị hủy để đảm bảo không còn thông tin đăng nhập nào tồn tại.
  + **Thiết yếu vì:** Quản lý phiên cung cấp một phương tiện đáng tin cậy để xác định người dùng trong suốt quá trình tương tác với ứng dụng, duy trì trạng thái đăng nhập, và quan trọng nhất là thực thi cơ chế phân quyền, ngăn chặn truy cập trái phép vào các chức năng hoặc dữ liệu nhạy cảm.