A white rectangular frame with blue border

AI-generated content may be incorrect.**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

A blue logo with a ball and a black background

AI-generated content may be incorrect.

**BÁO CÁO CUỐI KỲ**

**MÔN BẢO MẬT WEB**

**WEBSITE NẾN THƠM**

**GIẢNG VIÊN HƯỚNG DẪN: THS. LÊ THỊ MINH CHÂU**

**CÁC THÀNH VIÊN THAM GIA NHÓM 13**

|  |  |
| --- | --- |
| NGUYỄN PHÚC BẢO | 22110285 |
| ĐỖ PHÚ LUÂN | 22110372 |
| HUỲNH MINH MẪN | 22110377 |
| ĐẶNG MINH NHẬT | 22110389 |
| VI QUỐC THUẬN | 22110006 |

**TP. HỒ CHÍ MINH, THÁNG 5 NĂM 2025**

A white rectangular frame with blue border

AI-generated content may be incorrect.

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

A blue logo with a ball and a black background

AI-generated content may be incorrect.

**BÁO CÁO CUỐI KỲ**

**MÔN BẢO MẬT WEB**

**WEBSITE NẾN THƠM**

**GIẢNG VIÊN HƯỚNG DẪN: THS. LÊ THỊ MINH CHÂU**

**CÁC THÀNH VIÊN THAM GIA NHÓM 13**

|  |  |
| --- | --- |
| NGUYỄN PHÚC BẢO | 22110285 |
| ĐỖ PHÚ LUÂN | 22110372 |
| HUỲNH MINH MẪN | 22110377 |
| ĐẶNG MINH NHẬT | 22110389 |
| VI QUỐC THUẬN | 22110006 |

**TP. HỒ CHÍ MINH, THÁNG 5 NĂM 2025**

**NHẬN XÉT CỦA GIẢNG VIÊN**

|  |
| --- |
| ……………, ngày …, tháng 05, năm 2025  **Người nhận xét**  **(**Ký tên và ghi rõ họ tên) |

**MỤC LỤC**

[PHẦN MỞ ĐẦU 4](#_Toc199274224)

[1. Giới thiệu đề tài 4](#_Toc199274225)

[2. Mục tiêu đề tài 4](#_Toc199274226)

[3. Đối tượng đề tài 4](#_Toc199274227)

[PHẦN NỘI DUNG 5](#_Toc199274228)

[CHƯƠNG 1: ĐẶC TẢ PHẦN MỀM 5](#_Toc199274229)

[​1.​1. Thiết kế use case diagram 5](#_Toc199274230)

[​1.​2. Đặc tả use case 5](#_Toc199274231)

[1.​3 Đặc tả actor 7](#_Toc199274232)

[CHƯƠNG 2: THIẾT KẾ CƠ SỞ DỮ LIỆU 9](#_Toc199274233)

[2.1. Môhình diagram 9](#_Toc199274234)

[2.2. Cấu trúc cơ sở dữ liệu 9](#_Toc199274235)

[2.3. Bảng mô tả các trường trong bảng USER 11](#_Toc199274236)

[2.4. Bảng mô tả các trường trong ACCOUNT 11](#_Toc199274237)

[2.5. Bảng mô tả các trường trong ROLE 11](#_Toc199274238)

[2.6. Bảng mô tả các trường trong USERROLE 12](#_Toc199274239)

[2.7. Bảng mô tả các trường trong PRODUCT 12](#_Toc199274240)

[2.8. Bảng mô tả các trường trong CATEGORIES 12](#_Toc199274241)

[2.9. Bảng mô tả các trường trong PRODUCTCATEGORIES 13](#_Toc199274242)

[2.10. Bảng mô tả các trường trong CART 13](#_Toc199274243)

[2.11. Bảng mô tả các trường trong ORDERS 13](#_Toc199274244)

[2.12. Bảng mô tả các trường trong ORDERDETAILS 14](#_Toc199274245)

[2.13. Bảng mô tả các trường trong DISCOUNTS 14](#_Toc199274246)

[2.14. Bảng mô tả các trường trong REVIEW 15](#_Toc199274247)

[CHƯƠNG 3: THIẾT KẾ GIAO DIỆN 16](#_Toc199274248)

[3.1. Trang chủ 16](#_Toc199274249)

[3.2. Màn hình Login 17](#_Toc199274250)

[3.3. Màn hình Register 17](#_Toc199274251)

[3.4. Màn hình sản phẩm 18](#_Toc199274252)

[3.5. Màn hình chi tiết sản phẩm 18](#_Toc199274253)

[3.6. Màn hình Contact and Policy 19](#_Toc199274254)

[3.7. Màn hình giỏ hàng 20](#_Toc199274255)

[3.8. Màn hình cửa sổ ưu đãi 20](#_Toc199274256)

[3.9. Màn hình hồ sơ cá nhân 21](#_Toc199274257)

[3.10. Màn hình quản lý Products 21](#_Toc199274258)

[3.11. Màn hình thêm Product 22](#_Toc199274259)

[3.12. Màn hình sửa Product 22](#_Toc199274260)

[3.13. Màn hình quản lý hóa đơn 23](#_Toc199274261)

[3.14. Màn hình quản lý Ưu đãi 23](#_Toc199274262)

[3.15 Màn hình thêm Ưu đãi 24](#_Toc199274263)

[3.16. Màn hình chỉnh sửa ưu đãi 24](#_Toc199274264)

[3.17. Màn hình quản lý Danh mục 25](#_Toc199274265)

[3.18. Màn hình thêm Danh mục 25](#_Toc199274266)

[3.19. Màn hình chỉnh sửa Danh mục 26](#_Toc199274267)

[CHƯƠNG 4: CÀI ĐẶT MÔI TRƯỜNG 27](#_Toc199274268)

[4.1 Source code 27](#_Toc199274269)

[4.​2.​ Cài đặt Java 27](#_Toc199274270)

[​4.2.1.​ Download JDK 22 (Java Development Kit) 27](#_Toc199274271)

[​4.2.2.​ Thiết lập biến môi trường 27](#_Toc199274272)

[​4.3. Cài đặt server (Tomcat) 31](#_Toc199274273)

[​4.3.1.​ Download 31](#_Toc199274274)

[​4.3.2.​ Cấu hình Tomcat 31](#_Toc199274275)

[​4.4.​ DBMS (SQL Server 2019) 36](#_Toc199274276)

[​4.4.1.​ Download 36](#_Toc199274277)

[​4.4.2.​ Cài đặt 36](#_Toc199274278)

[​4.5.​ IDE for Java Web Developers (Eclipse) 39](#_Toc199274279)

[CHƯƠNG 5: CHẠY CHƯƠNG TRÌNH 40](#_Toc199274280)

[5.1. Chạy file script database 40](#_Toc199274281)

[5.2. Chỉnh sửa ConnectionUtil 40](#_Toc199274282)

[5.3. Import project vào eclipse 41](#_Toc199274283)

[5.4. Chạy chương trình 41](#_Toc199274284)

[5.5. Tài khoản tạo sẵn 41](#_Toc199274285)

[CHƯƠNG 6: BẢO MẬT WEB 43](#_Toc199274286)

[6.1. Kiểm tra bằng ZAP 43](#_Toc199274287)

[6.1.1. Absence of Anti-CSRF Tokens 43](#_Toc199274288)

[6.1.1.1. Khái niệm 43](#_Toc199274289)

[6.1.1.2. Tấn công project 43](#_Toc199274290)

[6.1.1.3. Nguyên nhân 44](#_Toc199274291)

[6.1.1.4. Khắc phục 44](#_Toc199274292)

[6.1.2. Content Security Policy (CSP) Header Not Set 47](#_Toc199274293)

[6.1.2.1. Khái niệm 47](#_Toc199274294)

[6.1.2.2. Tấn công project 47](#_Toc199274295)

[6.1.2.3. Nguyên nhân 48](#_Toc199274296)

[6.1.2.4. Khắc phục 49](#_Toc199274297)

[6.1.3. Cross-Domain Misconfiguration 52](#_Toc199274298)

[6.1.3.1. Khái niệm 52](#_Toc199274299)

[6.1.3.2. Tấn công Project 52](#_Toc199274300)

[6.1.3.3. Nguyên nhân 53](#_Toc199274301)

[6.1.3.4. Khắc phục 54](#_Toc199274302)

[6.1.4. Missing Anti-clickjacking Header 56](#_Toc199274303)

[6.1.4.1. Khái niệm 56](#_Toc199274304)

[6.1.4.2. Tấn công Project 56](#_Toc199274305)

[6.1.4.3. Nguyên nhân 57](#_Toc199274306)

[6.1.4.4. Khắc phục 58](#_Toc199274307)

[6.1.5. Session ID in URL Rewrite 59](#_Toc199274308)

[6.1.5.1. Khái niệm 59](#_Toc199274309)

[6.1.5.2. Tấn công 60](#_Toc199274310)

[6.1.5.3. Nguyên nhân 60](#_Toc199274311)

[6.1.5.4. Khắc phục 60](#_Toc199274312)

[6.1.6. Cookie without SameSite Attribute 62](#_Toc199274313)

[6.1.6.1. Khái niệm 62](#_Toc199274314)

[6.1.6.2. Tấn công project 63](#_Toc199274315)

[6.1.6.3. Nguyên nhân 63](#_Toc199274316)

[6.1.6.3. Giải pháp 64](#_Toc199274317)

[6.1.6.4. Khắc phục 65](#_Toc199274318)

[6.1.7. X-Content-Type-Options Header Missing 67](#_Toc199274319)

[6.1.7.1. Khái niệm 67](#_Toc199274320)

[6.1.7.2. Tấn công project 67](#_Toc199274321)

[6.1.7.3. Nguyên nhân 68](#_Toc199274322)

[6.1.7.4. Khắc phục 69](#_Toc199274323)

[6.1.8. Cross Site Script (XSS) 70](#_Toc199274324)

[6.1.8.1. Khái niệm 70](#_Toc199274325)

[6.1.8.2. Tấn công project 71](#_Toc199274326)

[6.1.8.3. Nguyên nhân 72](#_Toc199274327)

[6.1.8.4. Khắc phục 73](#_Toc199274328)

[6.1.8.4.1. Kiểm soát và lọc đầu vào (Input Validation) 73](#_Toc199274329)

[6.1.8.4.2. Mã hóa đầu ra theo ngữ cảnh (Contextual Output Encoding) 77](#_Toc199274330)

[6.1.8.4.3. Không bao giờ chèn dữ liệu người dùng trực tiếp vào mã HTML/JS 78](#_Toc199274331)

[6.1.8.4.4. Sử dụng thư viện bảo mật 78](#_Toc199274332)

[6.1.8.4.5. Kiểm thử lại với công cụ OWASP ZAP 80](#_Toc199274333)

[6.2. Kiểm tra thủ công 81](#_Toc199274334)

[6.2.1. Brute Force Attack 81](#_Toc199274335)

[6.2.1.1. Khái niệm 81](#_Toc199274336)

[6.2.1.2. Tấn công 81](#_Toc199274337)

[6.2.1.3. Nguyên nhân 88](#_Toc199274338)

[6.2.1.4. Khắc phục 88](#_Toc199274339)

[6.2.1.4.1 Góc độ nhà phát triển 88](#_Toc199274340)

[6.2.1.4.2. Góc độ người dùng: 93](#_Toc199274341)

[6.2.2 Broken Access Control 94](#_Toc199274342)

[6.2.2.1. Khái niệm 94](#_Toc199274343)

[6.2.2.2. Tấn công 95](#_Toc199274344)

[6.2.2.3. Nguyên nhân 96](#_Toc199274345)

[6.2.2.4. Khắc phục 96](#_Toc199274346)

[6.2.3. Buffer Overflow 98](#_Toc199274347)

[6.2.3.1. Khái niệm 98](#_Toc199274348)

[6.2.3.2. Tấn công 98](#_Toc199274349)

[6.2.3.3. Nguyên nhân 100](#_Toc199274350)

[6.2.3.4. Khắc phục 100](#_Toc199274351)

[6.2.4. Improper Input Validation 103](#_Toc199274352)

[6.2.4.1. Khái niệm 103](#_Toc199274353)

[6.2.4.2. Tấn công 104](#_Toc199274354)

[6.2.4.3. Nguyên nhân 106](#_Toc199274355)

[6.2.4.4. Khắc phục 106](#_Toc199274356)

[6.2.5. Vulnerable and Outdated Components 110](#_Toc199274357)

[6.2.5.1. Khái niệm 110](#_Toc199274358)

[6.2.5.2. Kiểm tra 110](#_Toc199274359)

[6.2.5.3. Nguyên nhân 113](#_Toc199274360)

[6.2.5.4. Khắc phục 113](#_Toc199274361)

[6.2.6. Insecure Client-Side Validation / Parameter Tampering 115](#_Toc199274362)

[6.2.6.1. Khái niệm 115](#_Toc199274363)

[6.2.6.2. Tấn công 116](#_Toc199274364)

[6.2.6.3. Nguyên nhân 118](#_Toc199274365)

[6.2.6.4. Khắc phục 119](#_Toc199274366)

[6.3. Các lỗi không phát hiện 121](#_Toc199274367)

[6.3.1. Insecure Direct Object Reference (IDOR) 121](#_Toc199274368)

[6.3.1.1. Khái niệm 121](#_Toc199274369)

[6.3.1.2. Xác định dấu hiệu 122](#_Toc199274370)

[6.3.1.3. Tiến hành kiểm tra 126](#_Toc199274371)

[6.3.2. SQL Injection 127](#_Toc199274372)

[6.3.2.1. Khái niệm 127](#_Toc199274373)

[6.3.2.2. Tấn công 128](#_Toc199274374)

[6.3.2.2.1. Thực hiện quét bằng ZAP 128](#_Toc199274375)

[6.3.2.2.2. Thực hiện quét bằng SQLMAP 129](#_Toc199274376)

[6.3.2.2.3. Thực hiện quét bằng jSQL Injection 130](#_Toc199274377)

[6.3.2.2.4. Thực hiện quét thủ công bằng tay 131](#_Toc199274378)

[6.3.2.2.5. Nguyên nhân 133](#_Toc199274379)

[6.4. Nâng cấp lên Https 136](#_Toc199274380)

[6.4.1. Giao thức HTTP là gì? 136](#_Toc199274381)

[6.4.2. Giao thức HTTPS là gì? 136](#_Toc199274382)

[6.4.3. Sự khác nhau giữa HTTP và HTTPS 137](#_Toc199274383)

[6.4.3.1. Mục đích sử dụng 137](#_Toc199274384)

[6.4.3.2. Chứng chỉ bảo mật SSL 137](#_Toc199274385)

[6.4.3.3. Port trên HTTP và HTPPS 137](#_Toc199274386)

[6.4.3.4. Mức độ bảo mật 138](#_Toc199274387)

[6.4.4. Thiết lập HTTPS 138](#_Toc199274388)

[CHƯƠNG 7: TỔNG HỢP LỖI 148](#_Toc199274389)

[PHẦN KẾT LUẬN 149](#_Toc199274390)

[TÀI LIỆU THAM KHẢO 150](#_Toc199274391)

# DANH MỤC HÌNH ẢNH

[Hình 1.1. Use case diagram 6](#_Toc199278199)

[Bảng 1.2. Đặc tả actor 8](#_Toc199278200)

[Hình 2.1. Mô hình diagram 10](#_Toc199278201)

[Bảng 2.1. Cấu trúc cơ sở dữ liệu 10](#_Toc199278202)

[Hình 3.1. Giao diện trang chủ 17](#_Toc199278203)

[Hình 3.2. Giao diện Login 18](#_Toc199278204)

[Hình 3.3. Giao diện Register 18](#_Toc199278205)

[Hình 3.4. Giao diện danh sách sản phẩm 19](#_Toc199278206)

[Hình 3.5. Giao diện chi tiết sản phẩm 19](#_Toc199278207)

[Hình 3.6. Giao diện Contact and Policy 20](#_Toc199278208)

[Hình 3.7. Giao diện Register 21](#_Toc199278209)

[Hình 3.8. Giao diện cửa sổ ưu đãi 21](#_Toc199278210)

[Hình 3.9. Giao diện hồ sơ cá nhân 22](#_Toc199278211)

[Hình 3.10. Giao diện quản lý Products 22](#_Toc199278212)

[Hình 3.11. Giao diện thêm Product 23](#_Toc199278213)

[Hình 3.12. Giao diện sửa Product 23](#_Toc199278214)

[Hình 3.13. Giao diện quản lý hóa đơn 24](#_Toc199278215)

[Hình 3.14. Giao diện quản lý ưu đãi 24](#_Toc199278216)

[Hình 3.15. Giao diện thêm Ưu đãi 25](#_Toc199278217)

[Hình 3.16. Giao diện sửa Ưu đãi 25](#_Toc199278218)

[Hình 3.17. Giao diện quản lý Danh mục 26](#_Toc199278219)

[Hình 3.18. Giao diện thêm danh mục 26](#_Toc199278220)

[Hình 3.19. Giao diện chỉnh sửa Danh mục 27](#_Toc199278221)

[Hình 4.1. Thuộc tính My Computer 28](#_Toc199278222)

[Hình 4.2. Giao diện setting Computer 29](#_Toc199278223)

[Hình 4.3. Giao diện System Properties Computer 29](#_Toc199278224)

[Hình 4.4. Giao diện Enviroment Variables Computer 30](#_Toc199278225)

[Hình 4.5. Giao diện thêm System Variable 30](#_Toc199278226)

[Hình 4.6. Giao diện đã thêm Enviroment Variable 31](#_Toc199278227)

[Hình 4.7. Giao diện chỉnh sửa Enviroment Variable 31](#_Toc199278228)

[Hình 4.8. Giao diện cài đặt Tomcat 32](#_Toc199278229)

[Hình 4.9. Nơi lưu trữ file startup.bat của tomcat 33](#_Toc199278230)

[Hình 4.10. Giao diện chạy startup.bat 33](#_Toc199278231)

[Hình 4.11. Giao diện chạy tomcat localhost 34](#_Toc199278232)

[Hình 4.12. Vị trí đổi port 34](#_Toc199278233)

[Hình 4.13. Giao diện Eclipse 35](#_Toc199278234)

[Hình 4.14. Giao diện Preferences Eclipse 35](#_Toc199278235)

[Hình 4.15. Giao diện chọn server apache Tomcat 36](#_Toc199278236)

[Hình 4.16. Giao diện chọn server Tomcat 36](#_Toc199278237)

[Hình 4.17. Giao diện server Tomcat 37](#_Toc199278238)

[Hình 4.18. File SQL đã tải về 38](#_Toc199278239)

[Hình 4.19. Giao diện chạy SQL .exe 38](#_Toc199278240)

[Hình 4.20. Giao diện cài đặt SQL server 39](#_Toc199278241)

[Hình 4.21. Giao diện cài đặt SQL server thành công 39](#_Toc199278242)

[Hình 5.1. Giao diện cây thư mục chứa ConnectioonUtil 41](#_Toc199278243)

[Hình 5.2. Code của file ConnectioonUtil 41](#_Toc199278244)

[Hình 5.3. Import project vào eclipse 42](#_Toc199278245)

[Bảng 5.4. Dữ liệu tài khoản sẵn có 42](#_Toc199278246)

[Hình 6.1. Lỗi Absence of Anti-CSRF Tokens phát hiện bởi ZAP 44](#_Toc199278247)

[Hình 6.2. Form bị lỗi Absence of Anti-CSRF Tokens phát hiện bởi ZAP 45](#_Toc199278248)

[Hình 6.3. Code file CSRFUtil 46](#_Toc199278249)

[Hình 6.4. Code file Login\_Servlet dùng attachToken 46](#_Toc199278250)

[Hình 6.5. Code file login.jsp dùng csrfToken 47](#_Toc199278251)

[Hình 6.6. Code file Login\_Servlet dùng isValid token 47](#_Toc199278252)

[Hình 6.7. Lỗi Absence of Anti-CSRF Tokens đã khắc phục 47](#_Toc199278253)

[Hình 6.8. Lỗi Content Security Policy (CSP) Header Not Set tìm thấy bởi ZAP 48](#_Toc199278254)

[Hình 6.9. Form login gây ra lỗi CSP 49](#_Toc199278255)

[Hình 6.10. Form register gây ra lỗi CSP 49](#_Toc199278256)

[Hình 6.11. Cài đặt CSP cho login.jsp và register.jsp 51](#_Toc199278257)

[Hình 6.12. Đoạn code Login.jsp 52](#_Toc199278258)

[Hình 6.13. Đoạn code Register.jsp 53](#_Toc199278259)

[Hình 6.14. Lỗi CSP đã được khắc phục 53](#_Toc199278260)

[Hình 6.15. Lỗi Cross-Domain Misconfiguration phát hiện bởi ZAP 54](#_Toc199278261)

[Hình 6.16. File code sử dụng all.min.css gây lỗi 54](#_Toc199278262)

[Hình 6.17. File code sử dụng toastr.min.css gây lỗi 54](#_Toc199278263)

[Hình 6.18. Thêm thư viện vào file pom.xml 56](#_Toc199278264)

[Hình 6.19. Loại bỏ code trong file jsp lỗi 56](#_Toc199278265)

[Hình 6.20. Code thay thế trong file jsp lỗi 56](#_Toc199278266)

[Hình 6.21. Lỗi Cross-Domain Misconfiguration đã được khác phục 57](#_Toc199278267)

[Hình 6.22. Lỗi issing Anti-clickjacking Header phát hiện bởi ZAP 58](#_Toc199278268)

[Hình 6.23. Lỗi Missing Anti-clickjacking Header đã được khắc phục 60](#_Toc199278269)

[Hình 6.24. Lỗi Session ID in URL Rewrite phát hiện bởi ZAP 61](#_Toc199278270)

[Hình 6.25. Lỗi Session ID in URL Rewrite đã được khắc phục 63](#_Toc199278271)

[Hình 6.26. Lỗi Cookie without SameSite Attribute phát hiện bởi ZAP 64](#_Toc199278272)

[Hình 6.27. Chi tiết lỗi Cookie without SameSite Attribute 64](#_Toc199278273)

[Hình 6.28. Trình duyệt Chrome 65](#_Toc199278274)

[Hình 6.29. Trình duyệt FireFox 65](#_Toc199278275)

[Hình 6.30. File context trong .metadata của eclipse workspace 67](#_Toc199278276)

[Hình 6.31. Thiết lập thuộc tính SameSite cho tất cả cookie 67](#_Toc199278277)

[Hình 6.32. Lỗi Cookie without SameSite Attribute đã được khắc phục 68](#_Toc199278278)

[Hình 6.33. Lỗi X-Content-Type-Options Header Missing phát hiện bởi ZAP 69](#_Toc199278279)

[Hình 6.34. File StaticResourceHeaderFilter 70](#_Toc199278280)

[Hình 6.35. File SecurityHeaderFilter 71](#_Toc199278281)

[Hình 6.36. Lỗi X-Content-Type-Options Header Missing đã được khắc phục 71](#_Toc199278282)

[Hình 6.37. Lỗi Cross Site Script (XSS) phát hiện bởi ZAP 73](#_Toc199278283)

[Hình 6.38. File product.jsp trước và sau khi được sửa 79](#_Toc199278284)

[Hình 6.39. Lỗi Cross Site Script (XSS) đã được khắc phục 81](#_Toc199278285)

[Hình 6.40. Tấn công XSS thử lại 82](#_Toc199278286)

[Hình 6.41. Tìm kiếm username tồn tại 83](#_Toc199278287)

[Hình 6.42. Bắt gói tin đăng ký bằng ZAP 83](#_Toc199278288)

[Hình 6.43. dùng ZAP Fuzzer để dò username tồn tại 84](#_Toc199278289)

[Hình 6.44. dùng ZAP Fuzzer để dò username 85](#_Toc199278290)

[Hình 6.45. Add Payload để dò username 86](#_Toc199278291)

[Hình 6.46. Brute Force username 86](#_Toc199278292)

[Hình 6.47. Dùng ZAP quét gói tin 87](#_Toc199278293)

[Hình 6.48. Add Payload để tiến hành Brute Force 88](#_Toc199278294)

[Hình 6.49. Brute Force password 88](#_Toc199278295)

[Hình 6.50. Giao diện hiển thị login 91](#_Toc199278296)

[Hình 6.51. Bổ dung hàm kiểm tra mật khẩu form đăng ký 93](#_Toc199278297)

[Hình 6.52. Báo lỗi khi nhập mật khẩu sai 94](#_Toc199278298)

[Hình 6.53. Giao diện web Strong Password 95](#_Toc199278299)

[Hình 6.54. Giao diện admin khi truy cập không có quyền admin 96](#_Toc199278300)

[Hình 6.55. Giao diện admin thông báo lỗi không có quyền truy cập 99](#_Toc199278301)

[Hình 6.56. Giao diện profile 100](#_Toc199278302)

[Hình 6.57. Giao diện profile thông báo lỗi cập nhật 100](#_Toc199278303)

[Hình 6.58. Màn hình console trong eclipse thông báo lỗi 101](#_Toc199278304)

[Hình 6.59. Code file profile.jsp 101](#_Toc199278305)

[Hình 6.60. Code file UpdateProfile\_Servlet 102](#_Toc199278306)

[Hình 6.61. Tấn công Buffer Overflow ở trang profile 102](#_Toc199278307)

[Hình 6.62. Tấn công Buffer Overflow profile đã khắc phục 103](#_Toc199278308)

[Hình 6.63. Database bảng Users sau khi tấn công khắc phục Buffer Overflow 103](#_Toc199278309)

[Hình 6.64. Code admin.jsp tương tự 104](#_Toc199278310)

[Hình 6.65. Code AddProduct\_Servlet tương tự 104](#_Toc199278311)

[Hình 6.66. Giao diện đăng ký 105](#_Toc199278312)

[Hình 6.67. ZAP bắt gói tin đăng ký 106](#_Toc199278313)

[Hình 6.68. Quay lại giao diện đăng nhập 106](#_Toc199278314)

[Hình 6.69. Dùng ZAP để sửa đổi 107](#_Toc199278315)

[Hình 6.70. Đăng nhập thành công giao diện trang chủ 107](#_Toc199278316)

[Hình 6.71. Thêm thư viện vào file pom.xml 108](#_Toc199278317)

[Hình 6.72. Bổ sung model TaiKhoan 108](#_Toc199278318)

[Hình 6.73. Code Register\_Servlet 109](#_Toc199278319)

[Hình 6.74. Code register.jsp 109](#_Toc199278320)

[Hình 6.75. Giao diện ZAP bắt gói tin 110](#_Toc199278321)

[Hình 6.76. Thông báo lỗi 110](#_Toc199278322)

[Hình 6.77. Code bổ bung login.jsp 110](#_Toc199278323)

[Hình 6.78. Code thông báo lỗi cho login.jsp 110](#_Toc199278324)

[Hình 6.79. Thêm plugin file pom.xml 111](#_Toc199278325)

[Hình 6.80. Thêm plugin file pom.xml 112](#_Toc199278326)

[Hình 6.81. File dependency-check-report.html 113](#_Toc199278327)

[Hình 6.82. Mở file dependency-check-report.html 113](#_Toc199278328)

[Hình 6.83. File suppression.xml 115](#_Toc199278329)

[Hình 6.84. Các suppress cung cấp từ dependency-check 115](#_Toc199278330)

[Hình 6.85. Import file vào plugin 115](#_Toc199278331)

[Hình 6.86. File .html sau khi build thành công 116](#_Toc199278332)

[Bảng 7.1 Bảng tổng hợp lỗi 147](#_Toc199278333)

# DANH MỤC BẢNG

[Bảng 1.1. Đặc tả Use case 7](#_Toc199278925)

[Bảng 2.2. Mô tả các trường bảng USER 13](#_Toc199278926)

[Bảng 2.3. Mô tả các trường bảng ACCOUNT 13](#_Toc199278927)

[Bảng 2.4. Mô tả các trường bảng ROLE 13](#_Toc199278928)

[Bảng 2.5. Mô tả các trường bảng USERROLE 14](#_Toc199278929)

[Bảng 2.6. Mô tả các trường bảng PRODUCT 14](#_Toc199278930)

[Bảng 2.7. Mô tả các trường bảng CATEGORIES 14](#_Toc199278931)

[Bảng 2.8. Mô tả các trường bảng PRODUCTCATEGORIES 15](#_Toc199278932)

[Bảng 2.9. Mô tả các trường bảng CART 15](#_Toc199278933)

[Bảng 2.10. Mô tả các trường bảng ORDERS 15](#_Toc199278934)

[Bảng 2.11. Mô tả các trường bảng ORDERDETAILS 16](#_Toc199278935)

[Bảng 2.12. Mô tả các trường bảng DISCOUNTS 16](#_Toc199278936)

[Bảng 2.13. Mô tả các trường bảng REVIEW 17](#_Toc199278937)

[Bảng 7.1. Bảng phân công 148](#_Toc199278938)

[Bảng 7.2. Bảng tổng hợp lỗi 148](#_Toc199278939)

# PHẦN MỞ ĐẦU

## 1. Giới thiệu đề tài

Trong thời đại công nghệ phát triển mạnh mẽ, mua sắm trực tuyến đã trở thành xu hướng phổ biến. Nhằm tận dụng tiềm năng của thị trường này, đồ án “Website bán nến thơm” được xây dựng với mục tiêu cung cấp một nền tảng trực tuyến giúp người dùng dễ dàng tìm kiếm, lựa chọn và mua sắm nến thơm chất lượng, tạo không gian sống thư giãn và dễ chịu. Website không chỉ mang đến trải nghiệm mua sắm tiện lợi, an toàn mà còn tối ưu hóa các tính năng như tìm kiếm, giỏ hàng và thanh toán trực tuyến, đáp ứng nhu cầu ngày càng cao của khách hàng.

## 2. Mục tiêu đề tài

Mục tiêu của đồ án "Website bán nến thơm" là xây dựng một nền tảng mua sắm trực tuyến chuyên nghiệp và dễ sử dụng, giúp người dùng dễ dàng tìm kiếm, lựa chọn và mua sắm các sản phẩm nến thơm chất lượng. Đề tài hướng đến việc tối ưu hóa trải nghiệm người dùng thông qua giao diện trực quan, thân thiện và tính năng tiện lợi, giúp người tiêu dùng tiết kiệm thời gian và công sức trong việc tìm kiếm sản phẩm. Đồng thời, website cũng sẽ cung cấp các công cụ quản lý giỏ hàng, thanh toán trực tuyến an toàn và bảo mật, đảm bảo một quy trình mua sắm thuận tiện và tin cậy. Qua đó, đề tài không chỉ giúp phát triển nền tảng bán hàng trực tuyến mà còn nâng cao chất lượng dịch vụ và sự hài lòng của khách hàng.

## 3. Đối tượng đề tài

Mục tiêu của đề tài là xây dựng một website bán nến thơm, người dùng bao gồm: user và manager. Tương ứng với từng loại người dùng sẽ có các chức năng khác nhau.​

# PHẦN NỘI DUNG

# CHƯƠNG 1: ĐẶC TẢ PHẦN MỀM

## ​1.​1. Thiết kế use case diagram

**A diagram of a diagram

Description automatically generated with medium confidence**

Hình 1.1. Use case diagram

## ​1.​2. Đặc tả use case

Bảng 1.1. Đặc tả Use case

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên use case** | **Ý nghĩa** |
| 1 | Usecase Đăng ký tài khoản | Khách hàng có thể đăng ký tài khoản mới bằng các thông tin như địa chỉ email, số điện thoại, họ tên, tên đăng nhập, mật khẩu và xác nhận mật khẩu. |
| 2 | Usecase Đăng nhập | Người dùng có thể đăng nhập vào hệ thống bằng tài khoản và mật khẩu. Dựa trên quyền của tài khoản, hệ thống sẽ hiển thị các chức năng tương ứng |
| 3 | Usecase Tìm kiếm và lọc sản phẩm | Khách hàng có thể tìm kiếm sản phẩm nến thơm theo tên sản phẩm mong muốn. Ngoài ra, khách hàng có thể lọc sản phẩm theo các tiêu chí như thương hiệu, mức giá để dễ dàng tìm ra sản phẩm phù hợp. |
| 4 | Usecase Quản lý giỏ hàng và ưu đãi | Khách hàng có thể xem giỏ hàng hiện tại, thêm sản phẩm vào giỏ hàng, xóa sản phẩm hoặc thay đổi số lượng sản phẩm trong giỏ. Đồng thời, khách hàng có thể chọn ưu đãi, mã giảm giá và tiến hành thanh toán cho sản phẩm đã chọn. |
| 5 | Usecase Quản lý thông tin cá nhân | Khách hàng có thể xem và cập nhật thông tin cá nhân như tên, email, số điện thoại và mật khẩu khi cần thiết. |
| 6 | Usecase Xem lịch sử đơn hàng | Khách hàng có thể xem lại lịch sử các đơn hàng đã được thanh toán và trạng thái của từng đơn hàng. |
| 7 | Usecase Chọn phương thức thanh toán | Khách hàng có thể lựa chọn phương thức thanh toán phù hợp như PAYPAL, VNPAY, MOMO hoặc các hình thức khác để thực hiện giao dịch. |
| 8 | Usecase Quản lý địa chỉ nhận hàng | Khách hàng có thể cập nhật, thêm mới hoặc chỉnh sửa địa chỉ giao hàng trước khi thanh toán. |
| 9 | Usecase Quản lý sản phẩm | Admin có thể xem danh sách các sản phẩm trong hệ thống, quản lý thông tin sản phẩm bao gồm tên, mô tả, giá cả, số lượng, v.v. Đồng thời, Admin có thể thêm, xóa và cập nhật thông tin sản phẩm trong hệ thống. |
| 10 | Usecase Quản lý khách hàng | Admin có thể xem danh sách khách hàng và thông tin chi tiết của từng khách hàng trong hệ thống. |
| 11 | Usecase Quản lý đơn hàng | Admin có thể xem danh sách các đơn hàng và theo dõi trạng thái của từng đơn hàng để đảm bảo quá trình xử lý đơn hàng diễn ra chính xác. |
| 12 | Usecase Quản lý loại sản phẩm | Admin có thể xem và cập nhật danh sách các loại sản phẩm có trong cửa hàng. |
| 13 | Usecase Quản lý ưu đãi | Admin có thể xem danh sách các ưu đãi hiện có và cập nhật các thông tin liên quan đến ưu đãi. |

## 1.​3. Đặc tả actor

Bảng 1.2. Đặc tả actor

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên Actor** | **Ý nghĩa** |
| 1 | Manager | 1. Quản lý có thể quản lý các sản phẩm trong hệ thống bao gồm:   * Xem danh sách các sản phẩm. * Thêm sản phẩm. * Xóa sản phẩm. * Sửa sản phẩm.   2. Quản lý có thể quản lý các danh mục của sản phẩm trong hệ thống bao gồm:   * Thêm danh mục. * Sửa danh mục. * Xóa danh mục. * Xem danh sách danh mục.   3. Quản lý có thể quản lý các ưu đãi dành cho người dùng sử dụng hệ thống bao gồm:   * Thêm ưu đãi. * Sửa ưu đãi. * Xóa ưu đãi. * Xem danh sách ưu đãi.   4. Quản lý có thể quản lý các đơn hàng của người dùng trong hệ thống bao gồm:   * Xem danh sách đơn hàng.   5. Ngoài ra, quản lý cũng có thể đăng nhập và đăng xuất trong hệ thống. |
| 2 | User | 1. Người dùng có thể tự quản lý giỏ hàng trong hệ thống bao gồm:   * Thêm vào giỏ hàng. * Xóa giỏ hàng. * Xem giỏ hàng. * Thanh toán   2. Người dùng có thể tìm kiếm các sản phẩm phù hợp với nhu cầu cá nhân trong hệ thống.  3. Người dùng có thể xem thông tin chi tiết các sản phẩm trong hệ thống.  4. Người dùng có thể quản lý thông tin cá nhân của chính mình trong hệ thống.  5. Người dùng có thể xem lịch sử hóa đơn của chính mình trong hệ thống. |

# CHƯƠNG 2: THIẾT KẾ CƠ SỞ DỮ LIỆU

## 2.1. Môhình diagram

A screenshot of a computer

Description automatically generated

Hình 2.1. Mô hình diagram

## 2.2. Cấu trúc cơ sở dữ liệu

Bảng 2.1. Cấu *trúc* cơ sở dữ liệu

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên bảng** | **Mục đích của bảng trong chương trình** |
| 1 | USERS | Chứa thông tin người dùng. Bao gồm: Mã người dùng, mã tài khoản, email, số điện thoại, địa chỉ. |
| 2 | ACCOUNT | Chứa thông tin tài khoản đăng nhập. Bao gồm: Mã tài khoản, tên đăng nhập, mật khẩu. |
| 3 | ROLE | Chứa thông tin quyền của người dùng. Bao gồm: Mã quyền, mã code quyền, tên quyền. |
| 4 | USERROLE | Bảng trung gian giữa USERS và ROLE để xác định vai trò của người dùng. Bao gồm: Mã người dùng, mã quyền. |
| 5 | PRODUCTS | Chứa thông tin về sản phẩm. Bao gồm: Mã sản phẩm, tên, mô tả, giá, số lượng trong kho, ảnh sản phẩm (dạng base64). |
| 6 | CATEGORIES | Chứa thông tin danh mục sản phẩm. Bao gồm: Mã danh mục, tên danh mục, mô tả. |
| 7 | PRODUCTCATEGORIES | Bảng trung gian giữa PRODUCTS và CATEGORIES để xác định sản phẩm thuộc danh mục nào. Bao gồm: Mã sản phẩm, mã danh mục. |
| 8 | CART | Chứa thông tin giỏ hàng của người dùng. Bao gồm: Mã giỏ hàng, mã người dùng, mã sản phẩm, số lượng, ngày tạo. |
| 9 | ORDERS | Chứa thông tin đơn hàng. Bao gồm: Mã đơn hàng, mã người dùng, tổng giá trị, trạng thái đơn hàng, địa chỉ giao hàng, ngày tạo đơn, mã giảm giá. |
| 10 | ORDERDETAILS | Bảng trung gian giữa ORDERS và PRODUCTS để xác định sản phẩm nào thuộc đơn hàng nào. Bao gồm: Mã đơn hàng, mã sản phẩm, số lượng, giá tiền. |
| 11 | DISCOUNTS | Chứa thông tin mã giảm giá. Bao gồm: Mã giảm giá, tên giảm giá, loại giảm giá, giá trị giảm giá, ngày bắt đầu, ngày kết thúc, trạng thái hoạt động. |
| 12 | REVIEWS | Chứa thông tin đánh giá của người dùng về sản phẩm. Bao gồm: Mã đánh giá, mã sản phẩm, mã người dùng, điểm số đánh giá, nhận xét, ngày tạo. |

## 2.3. Bảng mô tả các trường trong bảng USER

Bảng 2.2. Mô tả các trường bảng USER

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | UserID | Mã người dùng |
| 2 | AccountID | Mã tài khoản |
| 3 | Email | Email |
| 4 | PhoneNumber | Số điện thoại |
| 5 | Address | Địa chỉ |

## 2.4. Bảng mô tả các trường trong ACCOUNT

Bảng 2.3. Mô tả các trường bảng ACCOUNT

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | AccountID | Mã tài khoản |
| 2 | Username | Tên đăng nhập |
| 3 | Password | Mật khẩu |

## 2.5. Bảng mô tả các trường trong ROLE

Bảng 2.4. Mô tả các trường bảng ROLE

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | RoleID | Mã quyền |
| 2 | Name | Tên quyền |
| 3 | Code |  |

## 2.6. Bảng mô tả các trường trong USERROLE

Bảng 2.5. Mô tả các trường bảng USERROLE

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | UserID | Mã người dùng |
| 2 | RoleID | Mã quyền |

## 2.7. Bảng mô tả các trường trong PRODUCT

Bảng 2.6. Mô tả các trường bảng PRODUCT

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | ProductID | Mã sản phẩm |
| 2 | Name | Tên sản phẩm |
| 3 | Description | Mô tả |
| 4 | Price | Giá |
| 5 | Stock | Số lượng tồn kho |
| 6 | ImageBase64 | Hình ảnh |

## 2.8. Bảng mô tả các trường trong CATEGORIES

Bảng 2.7. Mô tả các trường bảng CATEGORIES

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | CategoryID | Mã danh mục |
| 2 | Name | Tên danh mục |
| 3 | Description | Mô tả |

## 2.9. Bảng mô tả các trường trong PRODUCTCATEGORIES

Bảng 2.8. Mô tả các trường bảng PRODUCTCATEGORIES

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | ProductID | Mã sản phẩm |
| 2 | CategoryID | Mã danh mục |

## 2.10. Bảng mô tả các trường trong CART

Bảng 2.9. Mô tả các trường bảng CART

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | CartID | Mã giỏ hàng |
| 2 | UserID | Mã người dùng |
| 3 | ProductID | Mã sản phẩm |
| 4 | Quantity | Số lượng |
| 5 | CreatedAt | Ngày tạo |

## 2.11. Bảng mô tả các trường trong ORDERS

Bảng 2.10. Mô tả các trường bảng ORDERS

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | OrderID | Mã đơn hàng |
| 2 | UserID | Mã người dùng |
| 3 | TotalPrice | Tổng giá trị |
| 4 | OrderStatus | Trạng thái đơn hàng |
| 5 | ShippingAddress | Địa chỉ giao hàng |
| 6 | CreatedAt | Ngày tạo đơn |
| 7 | DiscountCode | Mã giảm giá |

## 2.12. Bảng mô tả các trường trong ORDERDETAILS

Bảng 2.11. Mô tả các trường bảng ORDERDETAILS

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | OrderID | Mã đơn hàng |
| 2 | ProductID | Mã sản phẩm |
| 3 | Quantity | Số lượng |
| 4 | Price | Giá |

## 2.13. Bảng mô tả các trường trong DISCOUNTS

Bảng 2.12. Mô tả các trường bảng DISCOUNTS

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | DiscountID | Mã giảm giá |
| 2 | DiscountName | Tên giảm giá |
| 3 | DiscountType | Loại giảm giá |
| 4 | DiscountValue | Giá trị giảm giá |
| 5 | StartDate | Ngày bắt đầu |
| 6 | EndDate | Ngày kết thúc |
| 7 | IsActive | Trạng thái hoạt động |

## 2.14. Bảng mô tả các trường trong REVIEW

Bảng 2.13. Mô tả các trường bảng REVIEW

|  |  |  |
| --- | --- | --- |
| **TT** | **Tên trường** | **Mục đích của trường trong chương trình** |
| 1 | ReviewID | Mã đánh giá |
| 2 | ProductID | Mã sản phẩm |
| 3 | UserID | Mã người dùng |
| 4 | Rating | Điểm số đánh giá |
| 5 | Comment | Nhận xét |
| 6 | CreatedAt | Ngày tạo |

# CHƯƠNG 3: THIẾT KẾ GIAO DIỆN

## 3.1. Trang chủ

|  |
| --- |
|  |

Hình 3.1. Giao diện trang chủ

## 3.2. Màn hình Login

|  |
| --- |
|  |

Hình 3.2. Giao diện Login

## 3.3. Màn hình Register

|  |
| --- |
|  |

Hình 3.3. Giao diện Register

## 3.4. Màn hình sản phẩm

|  |
| --- |
|  |

Hình 3.4. Giao diện danh sách sản phẩm

## 3.5. Màn hình chi tiết sản phẩm

|  |
| --- |
|  |

Hình 3.5. Giao diện chi tiết sản phẩm

## 3.6. Màn hình Contact and Policy

|  |
| --- |
|  |

Hình 3.6. Giao diện Contact and Policy

## 3.7. Màn hình giỏ hàng

|  |
| --- |
|  |

Hình 3.7. Giao diện Register

## 3.8. Màn hình cửa sổ ưu đãi

|  |
| --- |
|  |

Hình 3.8. Giao diện cửa sổ ưu đãi

## 3.9. Màn hình hồ sơ cá nhân

|  |
| --- |
|  |

Hình 3.9. Giao diện hồ sơ cá nhân

## 3.10. Màn hình quản lý Products

|  |
| --- |
|  |

Hình 3.10. Giao diện quản lý Products

## 3.11. Màn hình thêm Product

|  |
| --- |
|  |

Hình 3.11. Giao diện thêm Product

## 3.12. Màn hình sửa Product

|  |
| --- |
|  |

Hình 3.12. Giao diện sửa Product

## 3.13. Màn hình quản lý hóa đơn

|  |
| --- |
|  |

Hình 3.13. Giao diện quản lý hóa đơn

## 3.14. Màn hình quản lý Ưu đãi

|  |
| --- |
|  |

Hình 3.14. Giao diện quản lý ưu đãi

## 3.15 Màn hình thêm Ưu đãi

|  |
| --- |
|  |

Hình 3.15. Giao diện thêm Ưu đãi

## 3.16. Màn hình chỉnh sửa ưu đãi

|  |
| --- |
|  |

Hình 3.16. Giao diện sửa Ưu đãi

## 3.17. Màn hình quản lý Danh mục

|  |
| --- |
|  |

Hình 3.17. Giao diện quản lý Danh mục

## 3.18. Màn hình thêm Danh mục

|  |
| --- |
|  |

Hình 3.18. Giao diện thêm danh mục

## 3.19. Màn hình chỉnh sửa Danh mục

|  |
| --- |
|  |

Hình 3.19. Giao diện chỉnh sửa Danh mục

# CHƯƠNG 4: CÀI ĐẶT MÔI TRƯỜNG

## 4.1 Source code

Link git: <https://github.com/Huynh-Minh-Man-04042004/Lap_Trinh_Web.git>

## 4.​2.​ Cài đặt Java

* Để phát triển bất kỳ một ứng dụng nào trên nền tảng Java thì trước tiên cần cài đặt JDK. JDK là bộ công cụ cho việc phát triển ứng dụng bằng ngôn ngữ lập trình Java.
* Nếu đã cài đặt JDK thì có thể bỏ qua bước này

### ​4.2.1.​ Download JDK 22 (Java Development Kit)

Download JDK 22 [tại đây](https://download.oracle.com/java/22/archive/jdk-22.0.2_linux-aarch64_bin.tar.gz).

Sau đó cài đặt bình thường.

### ​4.2.2.​ Thiết lập biến môi trường

Bước 1: Nhấp chuột phải vào My Computer → Properties → Advanced system settings → Advanced → Environment Variables.

|  |
| --- |
|  |

Hình 4.1. Thuộc tính My Computer

|  |
| --- |
|  |

Hình 4.2. Giao diện setting Computer

|  |
| --- |
|  |

Hình 4.3. Giao diện System Properties Computer

Bước 2: Nhấn New ở phần System variables để tạo mới một biến môi trường có tên JAVA\_HOME.

|  |
| --- |
|  |

Hình 4.4. Giao diện Enviroment Variables Computer

Bước 3: nhập vào các giá trị sau:

Variable name: JAVA\_HOME

Variable value: đường dẫn tới thư mục cài đặt JDK. *Ví dụ: C:\Users\bao\OneDrive\Máy tính\jPackage\jdk-22\_windows-x64\_bin\jdk-22.0.2*

|  |
| --- |
|  |

Hình 4.5. Giao diện thêm System Variable

Bước 4: Cập nhật biến môi trường **path**

Thêm vào phía trước giá trị của biến môi trường **path**: **%JAVA\_HOME%\bin;**

|  |
| --- |
|  |

Hình 4.6. Giao diện đã thêm Enviroment Variable

|  |
| --- |
|  |

Hình 4.7. Giao diện chỉnh sửa Enviroment Variable

Hoàn tất việc cài đặt Java.

## ​4.3. Cài đặt server (Tomcat)

### ​4.3.1.​ Download

Phiên bản là Tomcat 10.1

Lưu ý: download file zip (Có thể ấn vô [tại đây](https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.36/bin/apache-tomcat-10.1.36-windows-x64.zip) để download).

|  |
| --- |
|  |

Hình 4.8. Giao diện cài đặt Tomcat

Giải nén Tomcat sau khi download.

### ​4.3.2.​ Cấu hình Tomcat

Bước 1: Khởi động Tomcat

* Vào thư mục ~\{Tomcat}\bin (lưu ý {Tomcat} ở đây là tên folder của thư mục Tomcat đã giải nén), chạy file file startup.bat, cửa sổ tương tự bên dưới sẽ xuất hiện.

|  |
| --- |
|  |

Hình 4.9. Nơi lưu trữ file startup.bat của tomcat

|  |
| --- |
|  |

Hình 4.10. Giao diện chạy startup.bat

* Kiểm tra xem Tomcat đã khởi động thành công hay chưa: mở Chrome (hoặc trình duyệt bất kỳ), gõ vào địa chỉ: <http://localhost:8088>. Nếu xuất hiện trang web bên dưới, Tomcat đã khởi động thành công.

|  |
| --- |
|  |

Hình 4.11. Giao diện chạy tomcat localhost

**Lưu ý**: *có thể thay đổi port nếu muốn.*

Mở file server.xml trong thư mục ~\{Tomcat}\conf và sửa lại cổng theo mong muốn

VD: đổi port từ 8088 thành 8080

|  |
| --- |
|  |

Hình 4.12. Vị trí đổi port

Sau đó khởi động lại Tomcat và vào vào địa chỉ http://localhost:8080 để kiểm tra xem trang web có được load ko.

Bước 2: “Khai báo" Tomcat với Eclipse

* Trên Eclipse, chọn Window/References

|  |
| --- |
|  |

Hình 4.13. Giao diện Eclipse

* Thêm mới một Runtime Server: Ấn nút Add

|  |
| --- |
|  |

Hình 4.14. Giao diện Preferences Eclipse

* Chọn kiểu WebServer là Tomcat 10.1 -> Ấn Next

|  |
| --- |
|  |

Hình 4.15. Giao diện chọn server apache Tomcat

* Nhập đường dẫn đến thư mục Tomcat vừa giải nén, sau đó click OK.

|  |
| --- |
|  |

Hình 4.16. Giao diện chọn server Tomcat

* Kết quả

|  |
| --- |
|  |

Hình 4.17. Giao diện server Tomcat

## ​4.4.​ DBMS (SQL Server 2019)

### ​4.4.1.​ Download

Download SQL Server 2019 [tại đây.](https://go.microsoft.com/fwlink/?linkid=866664&clcid=0x409&culture=en-us&country=us)

### ​4.4.2.​ Cài đặt

Giải nén file download, ta được file cài đặt sau:

|  |
| --- |
|  |

Hình 4.18. File SQL đã tải về

Bước 1: Chạy file .exe đã tải

Bước 2: Chọn Download Media

|  |
| --- |
|  |

Hình 4.19. Giao diện chạy SQL .exe

Bước 3: Thực hiện như hình dưới và ấn Download

|  |
| --- |
|  |

Hình 4.20. Giao diện cài đặt SQL server

Bước 4: Kết quả

|  |
| --- |
|  |

Hình 4.21. Giao diện cài đặt SQL server thành công

## ​4.5.​ IDE for Java Web Developers (Eclipse)

Download phiên bản [**Eclipse IDE for Java EE Developers**](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2024-12/R/eclipse-jee-2024-12-R-win32-x86_64.zip&mirror_id=1319). Phiên bản này hỗ trợ viết các ứng dụng web.

Sau khi download, giải nén và sử dụng trực tiếp, ko cần cài đặt. (một số phiên bản eclipse có thể yêu cầu cài đặt).

# CHƯƠNG 5: CHẠY CHƯƠNG TRÌNH

## 5.1. Chạy file script database

Trong thư mục Lap\_Trinh\_Web clone về từ link git hãy **chạy file ScriptNenThomDB.sql**

## 5.2. Chỉnh sửa ConnectionUtil

Chọn thư mục src *→ Truy cập ConnectioonUtil.java*

|  |
| --- |
|  |

Hình 5.1. Giao diện cây thư mục chứa ConnectioonUtil

*→ hãy sửa phần password lại tương ứng với password tài khoản sa của SQL server của bạn*

|  |
| --- |
|  |

Hình 5.2. Code của file ConnectioonUtil

## 5.3. Import project vào eclipse

Vào *File → Import →* Existing Maven Projects *→* Chọn đường dẫn như hình *→* Finish

|  |
| --- |
|  |

Hình 5.3. Import project vào eclipse

## 5.4. Chạy chương trình

Click chuột phải vào tên Project -> Run as -> Run on server -> Chọn server tomcat -> Finish.

## 5.5. Tài khoản tạo sẵn

Bảng 5.4. Dữ liệu tài khoản sẵn có

|  |  |  |
| --- | --- | --- |
| **Vai trò** | **username** | **password** |
| Khách hàng | u1 | u1 |
| u2 | u2 |
| Quản lý | m1 | m1 |
| m2 | m2 |

# CHƯƠNG 6: BẢO MẬT WEB

## 6.1. Kiểm tra bằng ZAP

### 6.1.1. Absence of Anti-CSRF Tokens

#### 6.1.1.1. Khái niệm

Việc không sử dụng Anti-CSRF Token trong ứng dụng web có thể dẫn đến một lỗ hổng bảo mật nghiêm trọng. Khi thiếu cơ chế bảo vệ này, ứng dụng dễ trở thành mục tiêu của các cuộc tấn công giả mạo yêu cầu từ trang chéo (CSRF - Cross-Site Request Forgery). Trong trường hợp đó, kẻ xấu có thể lợi dụng trình duyệt của người dùng để gửi các yêu cầu không hợp lệ đến hệ thống mà người dùng không hề hay biết, ví dụ như thực hiện các giao dịch tài chính hoặc thay đổi thông tin cá nhân.

Để phòng chống loại tấn công này, hệ thống nên tích hợp cơ chế Anti-CSRF Token. Cơ chế này hoạt động bằng cách sinh ra một token duy nhất sau khi người dùng đăng nhập, và token đó được gửi kèm theo mỗi yêu cầu đến máy chủ. Khi máy chủ nhận được yêu cầu, nó sẽ xác minh token đó với token đã lưu trước đó. Nếu không trùng khớp, yêu cầu sẽ bị từ chối.

Áp dụng Anti-CSRF Token là một biện pháp thiết yếu trong việc đảm bảo an toàn cho ứng dụng web, giúp hạn chế các hành vi tấn công và bảo vệ dữ liệu người dùng hiệu quả hơn.

#### 6.1.1.2. Tấn công project

Sau khi sử dụng ZAP để quét lỗi, thì ta thấy hệ thống trả về cho chúng ta các nới, các đường dẫn, nơi đang bị lỗi CSRF

|  |
| --- |
|  |

Hình 6.1. Lỗi Absence of Anti-CSRF Tokens phát hiện bởi ZAP

Các lỗi này xuất phát chủ yếu từ form trong trang đăng nhập của website

|  |
| --- |
|  |

Hình 6.2. Form bị lỗi Absence of Anti-CSRF Tokens phát hiện bởi ZAP

#### 6.1.1.3. Nguyên nhân

Lỗi CSRF thường xảy ra khi các biểu mẫu HTML không được tích hợp token chống CSRF. Đây là một kỹ thuật bảo mật quan trọng giúp ngăn ngừa các cuộc tấn công giả mạo yêu cầu từ trang chéo. Khi form thiếu token này, kẻ tấn công có thể lợi dụng điểm yếu đó để thực hiện các hành động trái phép thay cho người dùng, chẳng hạn như đánh cắp dữ liệu hoặc thay đổi thông tin cá nhân. Để bảo vệ hệ thống khỏi nguy cơ này, các form HTML cần được cấu hình để tự động chèn một CSRF token duy nhất cho mỗi phiên làm việc, từ đó xác thực các yêu cầu gửi đến máy chủ một cách an toàn.

#### 6.1.1.4. Khắc phục

Để khắc phục lỗi *Absence of Anti-CSRF Tokens*, hệ thống đã triển khai cơ chế tạo và kiểm tra mã CSRF trong Servlet và JSP.

Tạo lớp tiện ích CSRFUtil.java với mục đích tạo ra các token ngẫu nhiên để gắn vào mỗi biểu mẫu có nguy cơ bị tấn công CSRF. Lớp tiện ích này sẽ thực hiện hai nhiệm vụ chính: sinh token mới và kiểm tra tính hợp lệ của token khi xử lý yêu cầu.

|  |
| --- |
|  |

Hình 6.3. Code file CSRFUtil

Ví dụ xử lý CSRF token ở Login:

Truy cập vào servlet hiển thị form đăng nhập (Login\_Servlet) và thêm đoạn mã gọi phương thức attachToken(request) từ CSRFUtil trong phương thức doGet, để sinh và gắn token vào session cũng như request.

|  |
| --- |
|  |

Hình 6.4. Code file Login\_Servlet dùng attachToken

Tiếp theo, trong file login.jsp, ta bổ sung một thẻ <input type="hidden"> chứa giá trị csrfToken vào bên trong biểu mẫu. Token này sẽ được gửi kèm theo mỗi lần người dùng gửi thông tin đăng nhập.

|  |
| --- |
|  |

Hình 6.5. Code file login.jsp dùng csrfToken

Sau đó, tại phương thức doPost của servlet Login\_Servlet, ta thêm đoạn kiểm tra token bằng cách gọi CSRFUtil.isValid(request). Nếu token không hợp lệ hoặc không tồn tại, hệ thống sẽ trả về lỗi 403 và không xử lý tiếp yêu cầu đăng nhập.

|  |
| --- |
|  |

Hình 6.6. Code file Login\_Servlet dùng isValid token

Cách làm này giúp bảo vệ form đăng nhập khỏi các yêu cầu giả mạo từ bên ngoài, ngay cả khi người dùng đang đăng nhập hợp lệ.

Sau khi hoàn tất, ta tiến hành dùng ZAP để quét lại ứng dụng và xác nhận rằng lỗ hổng liên quan đến CSRF đã được khắc phục.

|  |
| --- |
|  |

Hình 6.7. Lỗi Absence of Anti-CSRF Tokens đã khắc phục

### 6.1.2. Content Security Policy (CSP) Header Not Set

#### 6.1.2.1. Khái niệm

Content Security Policy (CSP) là một biện pháp bảo mật được thiết kế nhằm ngăn chặn hoặc giảm thiểu các cuộc tấn công như Cross-Site Scripting (XSS) và các hành vi chèn mã độc vào trang web. CSP hoạt động bằng cách thiết lập các quy tắc rõ ràng về nguồn tài nguyên nào (ví dụ: JavaScript, hình ảnh, font chữ, v.v.) được phép tải và thực thi bởi trình duyệt khi người dùng truy cập vào trang web.

Lỗi "Content Security Policy (CSP) Header Not Set" xuất hiện khi trang web không cấu hình hoặc thiếu tiêu đề CSP trong phản hồi HTTP. Việc này có thể tạo điều kiện cho kẻ xấu thực hiện các cuộc tấn công XSS hoặc chèn mã độc, từ đó đánh cắp thông tin người dùng, làm thay đổi giao diện trang hoặc phát tán phần mềm độc hại.

#### 6.1.2.2. Tấn công project

Sau khi khởi chạy trang web trên **localhost**, chúng ta tiến hành đưa liên kết vào công cụ **OWASP ZAP** để thực hiện các kiểm tra tự động. Công cụ này sẽ mô phỏng các cuộc tấn công một cách an toàn, không gây ảnh hưởng đến hệ thống thật. Mục tiêu là phát hiện ra các lỗ hổng bảo mật tiềm ẩn trong ứng dụng. Nhờ đó, chúng ta đã phát hiện ra một cảnh báo liên quan đến **việc chưa thiết lập tiêu đề Content Security Policy (CSP Header Not Set)** như minh họa trong hình dưới đây:

|  |
| --- |
|  |

Hình 6.8. Lỗi Content Security Policy (CSP) Header Not Set tìm thấy bởi ZAP

Các trang robots.txt, favicon.ico và sitemap.xml là các tệp thông tin mà các trình duyệt web thường tự động yêu cầu khi truy cập một trang web.

Lỗ hổng này chủ yếu xảy ra ở các trang:

* Login

|  |
| --- |
|  |

Hình 6.9. Form login gây ra lỗi CSP

* Register

|  |
| --- |
|  |

Hình 6.10. Form register gây ra lỗi CSP

#### 6.1.2.3. Nguyên nhân

Công cụ ZAP đã đưa ra cảnh báo về lỗ hổng này do việc không thiết lập Content Security Policy (CSP) có thể dẫn đến nhiều rủi ro bảo mật nghiêm trọng, đặc biệt là tấn công XSS (Cross-Site Scripting) hoặc chèn mã độc vào nội dung trang. Khi CSP không được cấu hình, trình duyệt sẽ không có giới hạn rõ ràng về nguồn gốc của các tài nguyên được phép tải về như JavaScript, CSS, hình ảnh, font chữ, v.v. Điều này tạo điều kiện cho kẻ tấn công chèn mã độc vào trang web một cách dễ dàng.

Ngược lại, khi cấu hình đúng CSP, quản trị viên có thể kiểm soát chặt chẽ các nguồn tài nguyên mà trang web cho phép, qua đó ngăn ngừa hoặc giảm thiểu đáng kể khả năng khai thác lỗ hổng. Việc thiếu CSP hoặc cấu hình sai có thể khiến trang web trở thành mục tiêu lý tưởng cho các cuộc tấn công và gây ảnh hưởng lớn đến tính bảo mật của hệ thống và dữ liệu người dùng.

#### 6.1.2.4. Khắc phục

Để thiết lập CSP header cho trang web, chúng ta có thẻ sử dụng Meta Tag CSP. Trong Meta Tag CSP, các Directive (chỉ thị) cần được cấu hình để xác định các hành động cụ thể mà trình duyệt nên thực hiện đối với các loại tài nguyên như script, style, image, font, và frame, … Một số directive quan trọng và thường được sử dụng được liệt kê như dưới đây:

* default-src: Directive này xác định các nguồn mặc định mà từ đó trang web có thể tải xuống các tài nguyên như hình ảnh, JavaScript, font chữ, v.v.

o Ví dụ: default-src 'self' https://trusted-link.com; cho phép tải xuống tài nguyên từ chính trang web (self) và từ đường link đáng tin cậy ([https://trusted-link.com](https://trusted-link.com/)).

* script-src: Directive này xác định các nguồn mà từ đó trang web có thể tải xuống và thực thi mã JavaScript.

o Ví dụ: script-src 'self' https://trusted-javascript.com; cho phép thực thi JavaScript từ chính trang web (self) và từ một nguồn JavaScript đáng tin cậy ([https://trusted-javascript.com](https://trusted-javascript.com/)).

* style-src: Directive này xác định các nguồn mà từ đó trang web có thể tải xuống và áp dụng CSS.

o Ví dụ: style-src 'self' https://trusted-css.com; cho phép tải xuống và áp dụng CSS từ chính trang web (self) và từ một nguồn CSS đáng tin cậy ([https://trusted-css.com](https://trusted-css.com/)).

* img-src: Directive này xác định các nguồn mà từ đó trang web có thể tải xuống hình ảnh.
* Ví dụ: img-src 'self' data:; cho phép tải xuống hình ảnh từ chính trang web (self) và hình ảnh được nhúng trong mã HTML (data:).
* font-src: Directive này xác định các nguồn mà từ đó trang web có thể tải xuống font chữ.

o Ví dụ: font-src 'self' https://trusted-fonts.com; cho phép tải xuống font chữ từ chính trang web (self) và từ một nguồn font chữ đáng tin cậy ([https://trusted-fonts.com](https://trusted-fonts.com/)).

* form-action: Xác định nguồn cho các hành động của biểu mẫu HTML, như các truy vấn POST hoặc GET. Directive này giúp ngăn chặn các cuộc tấn công CSRF (Cross-Site Request Forgery) bằng cách chỉ cho phép các biểu mẫu gửi dữ liệu đến các nguồn đã được xác thực.
* connect-src: Directive này xác định các nguồn mà trang web có thể kết nối đến, ví dụ như để thực hiện các yêu cầu HTTP.

o Ví dụ: connect-src 'self' https://api.example.com; cho phép kết nối đến chính trang web (self) và đến API đáng tin cậy ([https://api.example.com](https://api.example.com/)).

Một điều cần lưu ý khi cấu hình CSP header là nên tránh gặp lỗi Wildcard Directive. Wildcard directive trong Content Security Policy (CSP) là một directive cho phép tải tài nguyên từ tất cả các nguồn, bao gồm cả nguồn không an toàn. Directive này thường được biểu diễn bằng ký tự \*. Thay vì sử dụng Wildcard (\*), hãy chỉ định rõ các nguồn tài nguyên cho phép được xử lí. Ví dụ:

* default-src 'self' example.com;
* script-src 'self';
* style-src 'self' https://trusted-css.com;

Tiến hành đặt CSP cho các file login.jsp, register.jsp.

Với nội dung:

|  |
| --- |
|  |

Hình 6.11. Cài đặt CSP cho login.jsp và register.jsp

Trong đó:

CSP header trên được thiết lập bằng một thẻ <meta> với thuộc tính http-equiv được sử dụng để đặt giá trị của một header HTTP. Header này chứa các directive CSP (Content Security Policy) để xác định các nguồn tài nguyên được phép trên trang web.

* style-src: Cho phép load tất cả CSS từ chính domain.
* script-src: Cho phép load JS từ chính domain.
* default-src 'self': Mọi thứ khác (hình ảnh, fonts, media...) mặc định chỉ từ chính domain.
* form-action 'self': Chỉ cho phép gửi form về chính domain.

Login.jsp

|  |
| --- |
|  |

Hình 6.12. Đoạn code Login.jsp

Register.jsp

|  |
| --- |
|  |

Hình 6.13. Đoạn code Register.jsp

Và quét lại ZAP: Lỗi đã được fix

|  |
| --- |
|  |

Hình 6.14. Lỗi CSP đã được khắc phục

### 6.1.3. Cross-Domain Misconfiguration

#### 6.1.3.1. Khái niệm

Cross-Domain Misconfiguration là một lỗ hổng bảo mật xảy ra khi máy chủ web được cấu hình cho phép các miền bên ngoài (third-party domains) gửi yêu cầu truy cập tài nguyên mà không kiểm soát nghiêm ngặt, thông qua cơ chế CORS (Cross-Origin Resource Sharing). CORS là một tính năng bảo mật của trình duyệt giúp giới hạn quyền truy cập tài nguyên giữa các nguồn gốc khác nhau (cross-origin), nhưng nếu cấu hình sai, nó có thể trở thành điểm yếu nghiêm trọng.

Cụ thể, khi tiêu đề HTTP Access-Control-Allow-Origin được đặt giá trị là \* (wildcard) hoặc không giới hạn rõ ràng miền nào được phép truy cập, mọi website bên ngoài đều có thể gửi yêu cầu đến máy chủ và nhận dữ liệu phản hồi. Điều này đặc biệt nguy hiểm nếu các tài nguyên phản hồi chứa thông tin nhạy cảm hoặc có thể bị lợi dụng để thực hiện các hành vi độc hại.

Việc cấu hình sai CORS không trực tiếp gây ra tấn công, nhưng nó mở cửa cho các bên thứ ba gửi yêu cầu và thu thập dữ liệu, phá vỡ chính sách Same-Origin Policy (SOP) – một trong những nguyên tắc bảo mật cốt lõi của trình duyệt hiện đại.

#### 6.1.3.2. Tấn công Project

Khi sử dụng công cụ OWASP ZAP (Zed Attack Proxy) để quét và đánh giá các điểm yếu bảo mật phổ biến, lỗ hổng Cross-Domain Misconfiguration được phát hiện, đây là một cảnh báo nổi bật có mức độ rủi ro ở mức trung bình (Medium), được ghi nhận tại nhiều tài nguyên được tải từ máy chủ CDN bên ngoài, cụ thể như:

* https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css
* https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js
* https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js

ZAP phát hiện rằng các tài nguyên này được tải từ các miền bên ngoài có cấu hình CORS cho phép mọi miền (Access-Control-Allow-Origin: \*). Cấu hình này có thể dẫn đến khả năng các website bên thứ ba tùy ý thực hiện các yêu cầu đọc dữ liệu từ máy chủ, ngay cả khi người dùng không xác thực.

|  |
| --- |
|  |

Hình 6.15. Lỗi Cross-Domain Misconfiguration phát hiện bởi ZAP

|  |
| --- |
|  |

Hình 6.16. File code sử dụng all.min.css gây lỗi

|  |
| --- |
|  |

Hình 6.17. File code sử dụng toastr.min.css gây lỗi

#### 6.1.3.3. Nguyên nhân

Lỗi Cross-Domain Misconfiguration được ZAP phát hiện trong quá trình kiểm thử là do một số tài nguyên được tải từ máy chủ bên thứ ba (CDN) có cấu hình CORS không an toàn. Cụ thể, các tài nguyên như jquery.min.js, toastr.min.js, và font-awesome.min.css được cung cấp từ cdnjs.cloudflare.com, nơi có thiết lập tiêu đề Access-Control-Allow-Origin: \*.

Mặc dù ứng dụng không trực tiếp cấu hình CORS theo cách này, nhưng việc sử dụng các tài nguyên bên ngoài có cấu hình mở như vậy vẫn có thể gây ra rủi ro bảo mật, đặc biệt khi dữ liệu hoặc nội dung từ các nguồn này bị thay đổi, lợi dụng hoặc bị chèn mã độc.

ZAP xác định rằng các tài nguyên này cho phép đọc từ bất kỳ miền nào mà không có kiểm tra xác thực, điều này tạo điều kiện cho các trang web giả mạo có thể truy xuất hoặc tương tác với nội dung của hệ thống theo cách không kiểm soát. Đây là lý do dẫn đến cảnh báo về cấu hình sai chính sách chia sẻ tài nguyên giữa các miền.

#### 6.1.3.4. Khắc phục

Để khắc phục lỗi Cross-Domain Misconfiguration trong bối cảnh ứng dụng tải các thư viện từ bên thứ ba (CDN), hệ thống cần được điều chỉnh nhằm loại bỏ sự phụ thuộc vào các tài nguyên có cấu hình CORS không an toàn. Cụ thể, các tài nguyên được tải từ các máy chủ CDN như cdnjs.cloudflare.com hiện đang trả về tiêu đề HTTP Access-Control-Allow-Origin: \*, cho phép mọi miền có thể gửi yêu cầu đọc dữ liệu mà không bị giới hạn.

Mặc dù lỗi cấu hình CORS không phát sinh từ phía máy chủ của ứng dụng, nhưng do các tài nguyên đó được sử dụng trực tiếp trong giao diện người dùng, công cụ OWASP ZAP vẫn xác định đây là một lỗ hổng tiềm ẩn. Để giải quyết triệt để và đảm bảo không còn bị cảnh báo, các tài nguyên như thư viện jQuery, Toastr, hoặc Font Awesome cần được tải về và lưu trữ trực tiếp trên máy chủ nội bộ của hệ thống.

Việc chuyển sang sử dụng các tài nguyên nội bộ giúp loại bỏ hoàn toàn các vấn đề liên quan đến cấu hình CORS từ phía thứ ba, đồng thời kiểm soát tốt hơn nội dung được đưa vào trang web. Đây là một phương pháp hiệu quả để duy trì sự tuân thủ theo chính sách cùng nguồn gốc (Same-Origin Policy) của trình duyệt và đáp ứng yêu cầu kiểm thử bảo mật từ các công cụ như OWASP ZAP. Để xử lý triệt để và đảm bảo hệ thống không bị cảnh báo trong các lần kiểm thử tiếp theo, nhóm đã áp dụng các bước khắc phục sau: thêm thư viện WebJars vào pom.xml như sau:

|  |
| --- |
|  |

Hình 6.18. Thêm thư viện vào file pom.xml

Sau khi thêm vào file pom, ta tiến hành thay thế các đường dẫn CDN trong các file JSP:  
Loại bỏ các đoạn sau:

|  |
| --- |
|  |

Hình 6.19. Loại bỏ code trong file jsp lỗi

Thay bằng:

|  |
| --- |
|  |

Hình 6.20. Code thay thế trong file jsp lỗi

Tương tự như vậy, ta thay ở tất cả các file báo lỗi, và sau đó dùng ZAP để quét lại -> Lỗi đã được fix:

|  |
| --- |
|  |

Hình 6.21. Lỗi Cross-Domain Misconfiguration đã được khác phục

### 6.1.4. Missing Anti-clickjacking Header

#### 6.1.4.1. Khái niệm

Missing Anti-clickjacking Header là một lỗ hổng bảo mật xảy ra khi phản hồi từ máy chủ web không bao gồm các tiêu đề HTTP nhằm ngăn chặn các cuộc tấn công Clickjacking. Clickjacking là một kỹ thuật trong đó kẻ tấn công đánh lừa người dùng bằng cách tải nội dung của một trang hợp lệ trong một thẻ <iframe> và đặt phía trên nó các yếu tố giao diện trong suốt để lừa người dùng tương tác một cách vô thức (ví dụ: nhấn nút “Xác nhận”, “Mua hàng”, v.v.).

Để phòng chống kỹ thuật này, trình duyệt hiện đại hỗ trợ hai phương pháp cấu hình tiêu đề HTTP:

* X-Frame-Options: Chỉ thị này kiểm soát khả năng cho phép trang web được hiển thị trong một khung (frame). Có ba tùy chọn:
  + DENY: Không cho phép hiển thị trong bất kỳ khung nào.
  + SAMEORIGIN: Chỉ cho phép hiển thị nếu khung đến từ cùng một nguồn gốc (same-origin).
  + ALLOW-FROM uri: Chỉ cho phép hiển thị từ một URI cụ thể (hiện đã không còn được các trình duyệt chính hỗ trợ rộng rãi).
* Content-Security-Policy (CSP) với directive frame-ancestors: Đây là phương pháp hiện đại và linh hoạt hơn. Directive frame-ancestors cho phép xác định rõ những nguồn nào có thể nhúng trang web hiện tại vào khung.

Nếu không có một trong hai tiêu đề này, trình duyệt sẽ không có cơ chế bảo vệ mặc định, tạo điều kiện để kẻ tấn công thực hiện các cuộc tấn công clickjacking nhằm thao túng hành vi của người dùng mà họ không nhận ra.

#### 6.1.4.2. Tấn công Project

Khi sử dụng công cụ OWASP ZAP (Zed Attack Proxy) để quét và đánh giá các điểm yếu bảo mật phổ biến, lỗ hổng Missing Anti-clickjacking Header đã được phát hiện. Đây là một cảnh báo có mức độ rủi ro trung bình (Medium), được ghi nhận tại nhiều tài nguyên trong hệ thống, cụ thể như:

* http://localhost:8080/NenthomWeb/
* http://localhost:8080/NenthomWeb/views/register.jsp
* http://localhost:8080/NenthomWeb/servlets/Login\_Servlet

ZAP xác định rằng các phản hồi HTTP từ những đường dẫn này không chứa tiêu đề bảo vệ chống lại clickjacking, cụ thể là thiếu tiêu đề X-Frame-Options hoặc không có chỉ thị frame-ancestors trong Content-Security-Policy. Điều này khiến cho các trang có thể bị nhúng trong <iframe> bởi các website bên ngoài mà không có biện pháp hạn chế, từ đó mở ra khả năng bị tấn công Clickjacking nhằm đánh lừa người dùng thực hiện các hành động trái ý muốn trên giao diện bị che phủ.

|  |
| --- |
|  |

Hình 6.22. Lỗi issing Anti-clickjacking Header phát hiện bởi ZAP

#### 6.1.4.3. Nguyên nhân

Lỗi Missing Anti-clickjacking Header được OWASP ZAP phát hiện trong quá trình kiểm thử là do các phản hồi HTTP từ máy chủ không bao gồm các tiêu đề bảo mật cần thiết như X-Frame-Options hoặc chỉ thị frame-ancestors trong Content-Security-Policy. Cụ thể, khi truy cập các trang như login.jsp, register.jsp, và các servlet xử lý xác thực, phản hồi trả về hoàn toàn thiếu cơ chế kiểm soát việc trang có thể được hiển thị trong khung (iframe) hay không.

Mặc dù ứng dụng không cố ý cho phép các trang bị nhúng trong iframe, việc không cấu hình các tiêu đề này dẫn đến trình duyệt không có cơ chế bảo vệ mặc định, từ đó tạo điều kiện cho các cuộc tấn công clickjacking. Kẻ tấn công có thể lợi dụng điều này để nhúng trang vào một website giả mạo, sau đó sử dụng các lớp giao diện trong suốt nhằm đánh lừa người dùng thực hiện hành động trái ý muốn (như nhấn nút, xác nhận thao tác...).

ZAP xác định rằng toàn bộ phản hồi từ hệ thống đều không chứa bất kỳ chỉ thị chống clickjacking nào, điều này vi phạm nguyên tắc bảo mật Same-Origin Policy (SOP) và là lý do chính dẫn đến cảnh báo "Missing Anti-clickjacking Header".

#### 6.1.4.4. Khắc phục

Sau khi tiến hành kiểm thử bảo mật bằng công cụ OWASP ZAP, hệ thống đã ghi nhận lỗ hổng Missing Anti-clickjacking Header, phản ánh việc thiếu các biện pháp bảo vệ chống lại tấn công clickjacking trong phản hồi từ máy chủ web. Đây là lỗ hổng phổ biến nhưng nghiêm trọng, có thể bị khai thác để lừa người dùng thực hiện các hành động trái ý muốn trên giao diện bị giả mạo hoặc che khuất.

Trong quá trình phân tích, nhóm nhận thấy các trang như login.jsp, register.jsp và các servlet xử lý xác thực không được trang bị tiêu đề HTTP bảo vệ như X-Frame-Options hoặc chỉ thị frame-ancestors trong Content-Security-Policy. Do đó, trình duyệt không có khả năng ngăn chặn trang bị nhúng trong iframe từ các nguồn khác, làm tăng nguy cơ clickjacking.

Để xử lý lỗ hổng này một cách triệt để và có thể mở rộng cho toàn bộ ứng dụng, ta cần xây dựng một Servlet Filter có tên ClickjackingProtectionFilter. Filter này có nhiệm vụ tự động thêm tiêu đề: X-Frame-Options: DENY vào tất cả các phản hồi HTTP được trả về từ hệ thống. Cấu hình DENY đảm bảo rằng không trang nào trong ứng dụng có thể bị nhúng dưới dạng iframe, kể cả từ cùng miền.

|  |
| --- |
| package filters;  import jakarta.servlet.\*;  import jakarta.servlet.annotation.WebFilter;  import jakarta.servlet.http.HttpServletResponse;  import java.io.IOException;  @WebFilter("/\*") // Áp dụng cho tất cả các request  public class ClickjackingProtectionFilter implements Filter {  @Override  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  throws IOException, ServletException {  HttpServletResponse httpResp = (HttpServletResponse) response;  // Thêm header ngăn chặn Clickjacking  httpResp.setHeader("X-Frame-Options", "DENY");  // Nếu bạn dùng CSP thay thế:  // httpResp.setHeader("Content-Security-Policy", "frame-ancestors 'none'");  chain.doFilter(request, response);  }  } |

Filter được triển khai dưới dạng annotation với @WebFilter("/\*"), đảm bảo áp dụng cho toàn bộ tài nguyên và loại bỏ nhu cầu can thiệp vào từng tệp JSP hay servlet riêng lẻ. Việc này không chỉ giúp giải quyết lỗi được ZAP phát hiện mà còn đảm bảo tính nhất quán và khả năng bảo trì trong các lần phát triển tiếp theo.

Sau khi áp dụng thay đổi, hệ thống được khởi động lại và kiểm thử lại bằng ZAP. Kết quả cho thấy lỗi "Missing Anti-clickjacking Header" đã không còn xuất hiện, xác nhận rằng biện pháp bảo vệ đã được thiết lập thành công.

|  |
| --- |
|  |

Hình 6.23. Lỗi Missing Anti-clickjacking Header đã được khắc phục

### 6.1.5. Session ID in URL Rewrite

#### 6.1.5.1. Khái niệm

Session ID in URL Rewrite là một kỹ thuật mà các ứng dụng web sử dụng để theo dõi phiên làm việc (session) của người dùng bằng cách gắn mã phiên (Session ID) trực tiếp vào đường dẫn URL. Ví dụ:

|  |
| --- |
| [*http://localhost:8080/NenthomWeb/css/all.min.css;jsessionid=38B0A1B007516979F0ECB88308E95859*](http://localhost:8080/NenthomWeb/css/all.min.css;jsessionid=38B0A1B007516979F0ECB88308E95859) |

Thông thường, Session ID được lưu trữ trong cookie, nhưng trong một số trường hợp (ví dụ: khi trình duyệt không hỗ trợ cookie), ứng dụng sẽ sử dụng URL rewrite để duy trì trạng thái phiên.

#### 6.1.5.2. Tấn công

Sau khi thực hiện chạy web trên localhost, ta bắt đầu thực hiện quét OWASP ZAP bằng các trang tương ứng để tìm các lỗ hổng, sau khi quét thì tìm được các lỗ hổng về Session ID in URL

|  |
| --- |
|  |

Hình 6.24. Lỗi Session ID in URL Rewrite phát hiện bởi ZAP

#### 6.1.5.3. Nguyên nhân

Tomcat đang sử dụng cơ chế URL Rewriting để truyền session ID trong một số trường hợp, thường là với tài nguyên tĩnh (CSS, JS) hoặc khi cookie bị vô hiệu hóa hoặc trình duyệt không gửi lại cookie.

#### 6.1.5.4. Khắc phục

Cấu hình Tomcat để chỉ cho phép session qua cookie (vô hiệu hóa hoàn toàn URL rewriting). Tiến hành tạo ServletContextListener để cấu hình chế độ session:

|  |
| --- |
| package listeners;  import jakarta.servlet.ServletContextEvent;  import jakarta.servlet.ServletContextListener;  import jakarta.servlet.annotation.WebListener;  import jakarta.servlet.SessionTrackingMode;  import java.util.Collections;  @WebListener  public class SessionTrackingDisabler implements ServletContextListener {  @Override  public void contextInitialized(ServletContextEvent sce) {  // Chỉ cho phép session bằng cookie  sce.getServletContext().setSessionTrackingModes(  Collections.singleton(SessionTrackingMode.COOKIE)  );  }  @Override  public void contextDestroyed(ServletContextEvent sce) {  // nothing  }  } |

Đảm bảo servlet container quét @WebListener, thêm đoạn sau vào web.xml

|  |
| --- |
| <session-config>  <session-timeout>15</session-timeout> <!-- phút -->  </session-config>  <listener>  <listener-class>listeners.SessionTrackingDisabler</listener-class>  </listener> |

Cập nhật lại đoạn code sau trong Login.servlet để regenerate session sau đăng nhập:

|  |
| --- |
| request.getSession().invalidate();  HttpSession session = request.getSession(true);  session.setAttribute("username", username);  session.setAttribute("userID", userID);  session.setAttribute("role", role); |

Sau khi đã cập nhật và sửa đổi tiến hành request project và build trên localhost, sau đó đó tiến hành quét lại bằng ZAP và nhận được kết quả sau:

|  |
| --- |
|  |

Hình 6.25. Lỗi Session ID in URL Rewrite đã được khắc phục

→ Đã không còn phát hiện các lỗ hổng liên quan tới Session ID in URL rewrite.

### 6.1.6. Cookie without SameSite Attribute

#### 6.1.6.1. Khái niệm

SameSite Cookie là thuộc tính thêm vào trong cookie để yêu cầu chủ sở hữu trang web ghi rõ ràng nhãn cookie của Web khác, nhờ đó chỉ có thể chia sẻ Cookie với các trang web này. Như vậy một website lừa đảo không thể giả mạo người dùng vì không thể lấy được Cookie.

Lợi ích của SameSite giúp cho việc chống tấn công CSRF dễ dàng và hiệu quả hơn. Người làm hệ thống sẽ giảm chi phí do không cần bổ sung tính năng tạo và tương tác thông qua token ở client và server. Hiệu năng của hệ thống được nâng cao khi không phải sinh và đối chiếu token

SameSite là một thuộc tính của Set-Cookie trong Http response header. Có 3 giá trị:

- SameSite = Strict: Khi thuộc tính SameSite được đặt là Strict, cookie sẽ không được gửi cùng với các request được bắt đầu bởi các trang web của bên thứ 3

- SameSite = Lax: Khi bạn đặt thuộc tính SameSite của cookie thành Lax, cookie sẽ được gửi cùng với GET request được tạo bởi bên thứ 3.

- SameSite = None: Cookie sẽ được gửi trong mọi ngữ cảnh. Trình duyệt gửi cookie với cả request giữa các trang web và cùng một trang web.

#### 6.1.6.2. Tấn công project

Sau khi tiến hành quét ứng dụng web bằng công cụ ZAP, hệ thống đã phát hiện và liệt kê các đường dẫn chứa lỗ hổng liên quan đến Cookie không có thuộc tính SameSite (Cookie without SameSite Attribute):

|  |
| --- |
|  |

Hình 6.26. Lỗi Cookie without SameSite Attribute phát hiện bởi ZAP

#### 6.1.6.3. Nguyên nhân

Lỗi Cookie without SameSite Attribute thường xuất phát từ việc thuộc tính SameSite của cookie chưa được khởi tạo.

|  |
| --- |
|  |

Hình 6.27. Chi tiết lỗi Cookie without SameSite Attribute

Cookie có tên **JSESSIONID** chưa được thiết lập giá trị cho thuộc tính SameSite. Khi thuộc tính này không được chỉ định rõ ràng, các trình duyệt sẽ áp dụng giá trị mặc định khác nhau. Cụ thể:

* **Trên trình duyệt Chrome và Edge**, mặc dù không thiết lập SameSite, cookie sẽ được xử lý như thể đã đặt là 'Lax', nghĩa là chỉ được gửi trong các yêu cầu điều hướng từ cùng nguồn hoặc GET liên kết chéo.
* **Trên Firefox**, nếu SameSite không được khai báo, cookie sẽ mặc định là 'None', cho phép gửi trong mọi yêu cầu, kể cả từ các nguồn khác nhau — điều này tiềm ẩn nguy cơ bị khai thác trong các cuộc tấn công như CSRF.

|  |
| --- |
|  |

Hình 6.28. Trình duyệt Chrome

|  |
| --- |
|  |

Hình 6.29. Trình duyệt FireFox

#### 6.1.6.3. Giải pháp

Thiết lập SameSite cho cookie: Đảm bảo rằng khi tạo cookie, bạn thiết lập thuộc tính SameSite theo yêu cầu của ứng dụng web. Đặt SameSite thành "Strict", "Lax" hoặc "None" tùy thuộc vào yêu cầu cụ thể. Đối với các cookie cần được gửi trong yêu cầu giữa các trang web khác tên miền, hãy đặt SameSite thành "None" và đảm bảo rằng bạn cũng đặt thuộc tính "Secure" để chỉ ra rằng cookie chỉ được gửi qua các kết nối HTTPS bảo mật.

Cập nhật phiên bản trình duyệt: Đảm bảo rằng trình duyệt của bạn được cập nhật lên phiên bản mới nhất. Các phiên bản trình duyệt mới nhất thường có hỗ trợ tốt hơn cho thuộc tính SameSite và áp dụng hành vi mặc định nghiêm ngặt hơn. Điều này giúp đảm bảo rằng cookie được xử lý chính xác theo yêu cầu SameSite của bạn.

Kiểm tra cài đặt máy chủ: Đôi khi, lỗi Cookie không có thuộc tính SameSite có thể liên quan đến cấu hình máy chủ. Hãy kiểm tra cài đặt máy chủ của bạn và đảm bảo rằng nó không áp dụng các cài đặt không đúng hoặc không cấu hình đúng cho thuộc tính SameSite.

Sử dụng thư viện và framework cập nhật: Nếu bạn đang sử dụng các thư viện hoặc framework lập trình web, đảm bảo rằng bạn đang sử dụng phiên bản mới nhất và cập nhật của chúng. Phiên bản mới nhất thường đã được cập nhật để hỗ trợ thuộc tính SameSite và có các sửa lỗi liên quan đến cookie.

Kiểm tra và sửa lỗi mã nguồn: Kiểm tra mã nguồn ứng dụng web của bạn và đảm bảo rằng cookie được tạo ra và xử lý đúng cách. Tìm kiếm các vị trí mà cookie được tạo ra và đảm bảo rằng bạn đặt SameSite và các thuộc tính khác cần thiết cho cookie đó.

#### 6.1.6.4. Khắc phục

Đảm bảo rằng thuộc tính SameSite được đặt thành "Lax" hoặc lý tưởng nhất là "Strict" cho tất cả cookie.

Ta tiến hành mở file config/context.xml của tomcat trong eclipse ở thư mục .metadata.

<CookieProcessor className="org.apache.tomcat.util.http.Rfc6265CookieProcessor"

sameSiteCookies="Strict" />

|  |
| --- |
|  |

Hình 6.30. File context trong .metadata của eclipse workspace

để thiết lập thuộc tính SameSite là "Strict" cho tất cả các cookie.

|  |
| --- |
|  |

Hình 6.31. Thiết lập thuộc tính SameSite cho tất cả cookie

Đã thiết lập thuộc tính SameSite với giá trị là Strict.

Bắt đầu quét lại ZAP.

|  |
| --- |
|  |

Hình 6.32. Lỗi Cookie without SameSite Attribute đã được khắc phục

### 6.1.7. X-Content-Type-Options Header Missing

#### 6.1.7.1. Khái niệm

X-Content-Type-Options là một tiêu đề HTTP bảo mật được sử dụng để hướng dẫn trình duyệt không cố gắng tự suy đoán (sniff) loại nội dung của tài nguyên được tải về từ máy chủ, mà chỉ tin tưởng và xử lý tài nguyên dựa trên giá trị khai báo trong trường Content-Type. Việc thiết lập X-Content-Type-Options: nosniff giúp ngăn chặn các trình duyệt (đặc biệt là các phiên bản cũ như Internet Explorer hoặc Chrome cũ) diễn giải nhầm loại nội dung, điều có thể bị khai thác để thực thi mã độc hoặc gây sai lệch chức năng.

Nếu tiêu đề này không được đặt, kẻ tấn công có thể lợi dụng hành vi MIME-sniffing để chèn mã độc vào nội dung (ví dụ như tập tin .txt, .json hay .csv) và khiến trình duyệt xử lý như một tài nguyên thực thi (như HTML hoặc JavaScript). Điều này làm gia tăng nguy cơ tấn công kiểu XSS (Cross-site Scripting), từ đó có thể đánh cắp thông tin hoặc chiếm quyền điều khiển phiên làm việc.

#### 6.1.7.2. Tấn công project

Sau khi chạy web ở localhost ta đưa link vào ZAP để phần mềm này thực hiện các cuộc tấn công mà không gây hại cho web của mình, mục đích để tìm ra các lỗ hổng còn đang tồn tại, và nhờ vậy ta phát hiện lỗ hổng "X-Content-Type-Options Header Missing" trên nhiều phản hồi HTTP từ máy chủ nội bộ tại địa chỉ http://localhost:8080/NenthomWeb/.

Cụ thể, hệ thống tồn tại cảnh báo "X-Content-Type-Options Header Missing" tại nhiều điểm truy cập. Cụ thể, có tổng cộng 18 yêu cầu GET bị ảnh hưởng, bao gồm:

* Trang chủ ứng dụng: GET: http://localhost:8080/NenthomWeb/
* Các tài nguyên CSS tĩnh: styles\_header\_footer.css, styles\_Login.css (cả với và không có jsessionid)
* Tài nguyên động: GET/POST: /servlets/Login\_Servlet

Danh sách này cho thấy lỗ hổng không chỉ xảy ra ở các file CSS tĩnh mà còn xuất hiện trên cả servlet và JSP.

|  |
| --- |
|  |

Hình 6.33. Lỗi X-Content-Type-Options Header Missing phát hiện bởi ZAP

#### 6.1.7.3. Nguyên nhân

Lỗi "X-Content-Type-Options Header Missing" xảy ra khi máy chủ web không thiết lập tiêu đề X-Content-Type-Options: nosniff trong phản hồi HTTP. Khi thiếu tiêu đề này, trình duyệt có thể thực hiện MIME-sniffing và tự động diễn giải nội dung trả về dưới định dạng khác, dẫn đến nguy cơ bảo mật nếu tài nguyên bị sửa đổi hoặc bị kẻ xấu lợi dụng để chèn nội dung độc hại.

Nguyên nhân cụ thể bao gồm:

* Máy chủ (Tomcat) không gắn tiêu đề này theo mặc định cho các tài nguyên tĩnh như .css, .js.
* Các servlet, JSP hoặc controller không thiết lập thủ công tiêu đề này trong quá trình xử lý phản hồi.
* Các filter chưa áp dụng đầy đủ cho cả tài nguyên tĩnh và động, dẫn đến tiêu đề bị bỏ sót ở một số loại request nhất định.

#### 6.1.7.4. Khắc phục

Để sửa lỗi “X-Content-Type-Options Header Missing” do công cụ OWASP ZAP phát hiện, cần đảm bảo mọi phản hồi HTTP – bao gồm cả các tài nguyên tĩnh như CSS, JS, lẫn tài nguyên động như JSP, Servlet đều có tiêu đề X-Content-Type-Options: nosniff.

* Bảo vệ tài nguyên tĩnh: StaticResourceHeaderFilter

Một filter riêng được thiết lập với annotation @WebFilter(urlPatterns = {"\*.css", "\*.js"}), đảm bảo rằng tất cả các tệp CSS và JS được trả về từ máy chủ đều kèm theo tiêu đề X-Content-Type-Options: nosniff. Điều này giải quyết triệt để lỗi xuất hiện trên các URL như:

GET: /css/styles\_Login.css

GET: /css/styles\_header\_footer.css

|  |
| --- |
|  |

Hình 6.34. File StaticResourceHeaderFilter

* Bảo vệ tài nguyên động: SecurityHeaderFilter

Một filter thứ hai được áp dụng cho tất cả các request đến tài nguyên động như JSP, servlet, controller, … Cụ thể sử dụng @WebFilter("/\*") để bao phủ các endpoint như:

GET: /views/login.jsp

GET/POST: /servlets/Login\_Servlet

GET: /

|  |
| --- |
|  |

Hình 6.35. File SecurityHeaderFilter

Sau khi triển khai và khởi động lại máy chủ, thực hiện quét lại bằng ZAP cho thấy tất cả các phản hồi từ máy chủ bao gồm cả tài nguyên tĩnh và động đều đã được bổ sung tiêu đề X-Content-Type-Options, và cảnh báo bảo mật không còn xuất hiện.

|  |
| --- |
|  |

Hình 6.36. Lỗi X-Content-Type-Options Header Missing đã được khắc phục

### 6.1.8. Cross Site Script (XSS)

#### 6.1.8.1. Khái niệm

XSS (Cross-site Scripting) là một lỗ hổng phổ biến trong ứng dụng web. Để khai thác một lỗ hổng XSS, attacker sẽ chèn mã độc thông qua các đoạn script để thực thi chúng ở phía client. Thông thường, các cuộc tấn công XSS được sử dụng để vượt qua các kiểm soát truy cập và mạo danh người dùng. XSS thường có 3 loại: Reflected XSS, Stored XSS, DOM-based XSS.

Ở project này thì tấn công theo loại Stored XSS. Trước tiên, kẻ tấn công chèn đoạn mã độc vào các điểm đầu vào không được kiểm tra kỹ, như form, input, hoặc textarea, để lưu vào cơ sở dữ liệu. Sau đó, khi nạn nhân truy cập vào ứng dụng web và tương tác với dữ liệu đã được lưu này, đoạn mã của kẻ tấn công sẽ được thực thi trên trình duyệt của nạn nhân.

Sự khác biệt chính giữa Reflected XSS và Stored XSS là:

* Đối với Reflected XSS, kẻ tấn công cần phải lừa nạn nhân truy cập vào URL chứa mã độc. Trong khi đó, với Stored XSS, kẻ tấn công không cần phải thực hiện việc này, mà chỉ cần đợi nạn nhân tự động truy cập vào ứng dụng web, không hề hay biết rằng dữ liệu đã bị nhiễm độc.
* Mục tiêu của kẻ tấn công dễ dàng đạt được hơn nếu nạn nhân vẫn đang trong phiên làm việc của ứng dụng web. Với Reflected XSS, kẻ tấn công có thể cố gắng thuyết phục nạn nhân đăng nhập và truy cập vào URL chứa mã độc. Nhưng với Stored XSS, mỗi khi nạn nhân truy cập các chức năng liên quan, mã độc sẽ được thực thi, thường là những chức năng yêu cầu xác thực trước đó.

Stored XSS nguy hiểm hơn rất nhiều so với Reflected XSS, vì nó có thể ảnh hưởng đến tất cả người dùng của ứng dụng web đó. Đặc biệt, nếu nạn nhân có vai trò quản trị, nguy cơ bị chiếm quyền điều khiển toàn bộ trang web là rất cao.

#### 6.1.8.2. Tấn công project

Khi thực hiện kiểm thử bảo mật bằng OWASP ZAP, hệ thống đã phát hiện 4 lỗ hổng XSS tại các điểm sau:

**Stored XSS (Persistent)**

URL bị ảnh hưởng: GET <http://localhost:8080/NenthomWeb/servlets/DSProduct_Servlet?page=product>

Đây là dạng Stored XSS, nghĩa là dữ liệu độc hại đã được ghi nhận lại (có thể trong cơ sở dữ liệu hoặc session) và được hiển thị lại trên giao diện mà không được kiểm tra hay mã hóa đúng cách. Kẻ tấn công có thể chèn đoạn mã JavaScript vào một form nhập liệu (ví dụ: tên sản phẩm), mã này sau đó sẽ được hiển thị cho những người dùng khác khi họ truy cập trang DSProduct\_Servlet.

**Reflected XSS**

URL bị ảnh hưởng: POST <http://localhost:8080/NenthomWeb/servlets/AddProduct_Servlet>

Reflected XSS được phát hiện ở 3 truy vấn POST đến servlet AddProduct\_Servlet. OWASP ZAP phát hiện rằng khi gửi dữ liệu có chứa script độc hại trong các trường POST (ví dụ: tên sản phẩm, mô tả, giá), server phản hồi lại dữ liệu này trực tiếp lên trang mà không lọc hoặc mã hóa, dẫn đến việc đoạn mã được thực thi trên trình duyệt người dùng.

|  |
| --- |
|  |

Hình 6.37. Lỗi Cross Site Script (XSS) phát hiện bởi ZAP

#### 6.1.8.3. Nguyên nhân

Nguyên nhân chính của các lỗ hổng XSS trên là do dữ liệu đầu vào từ người dùng không được kiểm tra hoặc mã hóa đúng cách trước khi hiển thị lên giao diện. Cụ thể:

Ở **DSProduct\_Servlet**, hệ thống đang hiển thị thông tin sản phẩm từ cơ sở dữ liệu mà không escape các ký tự đặc biệt như <, >, ", ', script, document, alert, http…

|  |
| --- |
| ProductDAO productDao = **new** ProductDAO(connection);  List<Product> products = productDao.getAllProducts();  OrderDAO orderDao = **new** OrderDAO(connection);  List<Order> orders = orderDao.getAllOrders();  DiscountDAO discountDAO = **new** DiscountDAO();  List<Discount> discounts = discountDAO.getActiveDiscounts();    CategorieDAO categorieDAO = **new** CategorieDAO(connection);  List<Categorie> categories = categorieDAO.getAllCategories();  **if** ("admin".equals(page)) {  request.setAttribute("products", products);  request.setAttribute("orders", orders);  request.setAttribute("discounts", discounts);  request.setAttribute("categories", categories);  request.getRequestDispatcher("/views/admin.jsp").forward(request, response);  } **else** {  request.setAttribute("products", products);  request.setAttribute("categories", categories);  request.getRequestDispatcher("/views/product.jsp").forward(request, response);  } |

Ở **AddProduct\_Servlet**, các tham số được POST lên server như tên sản phẩm, mô tả, giá... được lưu thẳng vào trong database mà không kiểm tra nội dung hoặc mã hóa HTML an toàn.

|  |
| --- |
| String name = request.getParameter("name");  **double** price = Double.*parseDouble*(request.getParameter("price"));  String description = request.getParameter("description");  **int** stock = Integer.*parseInt*(request.getParameter("stock"));  String imageBase64 = request.getParameter("imageBase64");  Product product = **new** Product(0, name, description, price, stock, imageBase64);  ***logger***.info("Bắt đầu thêm sản phẩm: " + name + ", giá: " + price + ", tồn kho: " + stock);  **try** (Connection connection = ConnectionUtil.*DB*()) {  ProductDAO productDAO = **new** ProductDAO(connection);  productDAO.addProduct(product);    ***logger***.info("Thêm sản phẩm thành công: " + name);  response.sendRedirect("/NenthomWeb/servlets/DSProduct\_Servlet?page=admin&message=success&action=product");  } **catch** (Exception e) {  ***logger***.severe("Lỗi khi thêm sản phẩm: " + e.getMessage());  e.printStackTrace();  response.sendError(HttpServletResponse.***SC\_INTERNAL\_SERVER\_ERROR***, "Đã xảy ra lỗi khi kết nối với cơ sở dữ liệu.");  } |

Việc không áp dụng các biện pháp như escaping, encoding, input validation là nguyên nhân cốt lõi dẫn đến XSS.

#### 6.1.8.4. Khắc phục

Để xử lý triệt để các lỗ hổng XSS, cần thực hiện các biện pháp sau:

##### 6.1.8.4.1. Kiểm soát và lọc đầu vào (Input Validation)

Chỉ chấp nhận dữ liệu hợp lệ: kiểm tra định dạng, loại dữ liệu, độ dài, kiểu ký tự,... trước khi xử lý. Loại bỏ dữ liệu nguy hiểm: đặc biệt là các ký tự như <, >, ", ', &, /, hoặc bất kỳ đoạn mã JavaScript có thể được chèn.

Không tin tưởng bất kỳ dữ liệu nào từ người dùng, bao gồm URL, form, cookie, header,...

|  |
| --- |
| **try** {  String name = request.getParameter("name");  **if** (name != **null**) {  name = name.trim();  **if** (name.length() > 100) {  name = name.substring(0, 100);  }  }  **if** (!name.matches("[a-zA-Z0-9 ]{1,50}")) {  **throw** **new** IllegalArgumentException("Tên sản phẩm không hợp lệ. Vui lòng chỉ dùng ký tự chữ, số và khoảng trắng (tối đa 50 ký tự).");  }  **double** price = Double.*parseDouble*(request.getParameter("price"));  String description = request.getParameter("description");  **if** (description != **null**) {  description = description.trim();  **if** (description.length() > 1000) {  description = description.substring(0, 1000);  }  }  **if** (!description.matches("[a-zA-Z0-9 ]{1,500}")) {  **throw** **new** IllegalArgumentException("Mô tả sản phẩm không hợp lệ. Chỉ chấp nhận chữ, số, khoảng trắng (tối đa 500 ký tự).");  }  **int** stock = Integer.*parseInt*(request.getParameter("stock"));  String imageBase64 = request.getParameter("imageBase64");  Product product = **new** Product(0, name, description, price, stock, imageBase64);  ***logger***.info("Bắt đầu thêm sản phẩm: " + name + ", giá: " + price + ", tồn kho: " + stock);  **try** (Connection connection = ConnectionUtil.*DB*()) {  ProductDAO productDAO = **new** ProductDAO(connection);  productDAO.addProduct(product);  ***logger***.info("Thêm sản phẩm thành công: " + name);  response.sendRedirect("/NenthomWeb/servlets/DSProduct\_Servlet?page=admin&message=success&action=product");  }  } **catch** (IllegalArgumentException e) {  ***logger***.warning("Dữ liệu đầu vào không hợp lệ: " + e.getMessage());  request.setAttribute("errorMessage", e.getMessage());  request.getRequestDispatcher("/views/xss\_error.jsp").forward(request, response);  } **catch** (Exception e) {  ***logger***.severe("Lỗi khi thêm sản phẩm: " + e.getMessage());  e.printStackTrace();  response.sendError(HttpServletResponse.***SC\_INTERNAL\_SERVER\_ERROR***,  "Đã xảy ra lỗi khi kết nối với cơ sở dữ liệu.");  }  } |

|  |
| --- |
| **try** {  **int** categoryId = Integer.*parseInt*(request.getParameter("categoryID"));  String name = request.getParameter("name");  String description = request.getParameter("description");  **if** (name != **null**) {  name = name.trim();  **if** (name.length() > 50) {  name = name.substring(0, 50);  }  } **else** {  **throw** **new** IllegalArgumentException("Tên danh mục không được để trống.");  }  **if** (description != **null**) {  description = description.trim();  **if** (description.length() > 1000) {  description = description.substring(0, 1000);  }  } **else** {  description = "";  }  // Kiểm tra ký tự hợp lệ  **if** (!name.matches("[a-zA-Z0-9 \\-]{1,50}")) {  **throw** **new** IllegalArgumentException("Tên danh mục không hợp lệ. Chỉ được dùng chữ, số, khoảng trắng và dấu gạch ngang.");  }  **if** (!description.matches("[a-zA-Z0-9 ,.\\-]{0,1000}")) {  **throw** **new** IllegalArgumentException("Mô tả không hợp lệ. Chỉ được dùng ký tự chữ, số, dấu phẩy, chấm, khoảng trắng.");  }  // Kiểm tra từ khóa JavaScript nguy hiểm  **if** (XSSUtil.*containsBannedKeyword*(name) || XSSUtil.*containsBannedKeyword*(description)) {  **throw** **new** IllegalArgumentException("Dữ liệu không được chứa từ khóa JavaScript nguy hiểm.");  }  Categorie updatedCategory = **new** Categorie(categoryId, name, description); |
| **import utils.XSSUtil; // Import lớp thư viện để dùng** |

|  |
| --- |
| **package** utils;  **import** java.util.regex.Pattern;  **public** **class** XSSUtil {  // Danh sách từ khóa JS nguy hiểm cần cấm  **private** **static** **final** String[] ***BANNED\_KEYWORDS*** = {  "script", "onload", "onclick", "onerror", "onmouseover", "onmouseenter",  "onmouseleave", "javascript:", "eval(", "document.cookie", "alert("  };  // Regex chuẩn để kiểm tra chuỗi hợp lệ  // Tên danh mục (chữ, số, khoảng trắng, dấu gạch ngang)  **private** **static** **final** Pattern ***VALID\_NAME\_PATTERN*** = Pattern.*compile*("^[a-zA-Z0-9 \\-]{1,50}$");  // Mô tả (chữ, số, dấu phẩy, chấm, khoảng trắng)  **private** **static** **final** Pattern ***VALID\_DESCRIPTION\_PATTERN*** = Pattern.*compile*("^[a-zA-Z0-9 ,.\\-]{0,1000}$");  // DiscountName (chữ, số, dấu gạch ngang, dấu chấm, dấu phẩy, khoảng trắng)  **private** **static** **final** Pattern ***VALID\_DISCOUNT\_NAME\_PATTERN*** = Pattern.*compile*("^[a-zA-Z0-9 \\-.,]{1,255}$");  // Kiểm tra xem chuỗi có chứa từ khóa JS nguy hiểm không  **public** **static** **boolean** containsBannedKeyword(String input) {  **if** (input == **null**) **return** **false**;  String lower = input.toLowerCase();  **for** (String keyword : ***BANNED\_KEYWORDS***) {  **if** (lower.contains(keyword)) {  **return** **true**;  }  }  **return** **false**;  }  // Kiểm tra tên danh mục hợp lệ  **public** **static** **boolean** isValidCategoryName(String input) {  **if** (input == **null**) **return** **false**;  **return** ***VALID\_NAME\_PATTERN***.matcher(input).matches() && !*containsBannedKeyword*(input);  }  // Kiểm tra mô tả hợp lệ  **public** **static** **boolean** isValidCategoryDescription(String input) {  **if** (input == **null**) **return** **true**; // Mô tả có thể trống  **return** ***VALID\_DESCRIPTION\_PATTERN***.matcher(input).matches() && !*containsBannedKeyword*(input);  }  // Kiểm tra discountName hợp lệ  **public** **static** **boolean** isValidDiscountName(String input) {  **if** (input == **null**) **return** **false**;  **return** ***VALID\_DISCOUNT\_NAME\_PATTERN***.matcher(input).matches() && !*containsBannedKeyword*(input);  }  } |

##### 6.1.8.4.2. Mã hóa đầu ra theo ngữ cảnh (Contextual Output Encoding)

Escape khi dữ liệu trước khi respone:

|  |
| --- |
| ProductDAO productDao = **new** ProductDAO(connection);  List<Product> products = productDao.getAllProducts();  **for** (Product p : products) {  String escapedName = StringEscapeUtils.*escapeHtml4*(p.getName());  escapedName = StringEscapeUtils.*escapeEcmaScript*(escapedName);  p.setName(escapedName);    String escapedDecription = StringEscapeUtils.*escapeHtml4*(p.getDescription());  escapedDecription = StringEscapeUtils.*escapeEcmaScript*(escapedDecription);  p.setDescription(escapedDecription);  }  OrderDAO orderDao = **new** OrderDAO(connection);  List<Order> orders = orderDao.getAllOrders();  **for** (Order p : orders) {  String escapedShippingAddress = StringEscapeUtils.*escapeHtml4*(p.getShippingAddress());  escapedShippingAddress = StringEscapeUtils.*escapeEcmaScript*(escapedShippingAddress);  p.setShippingAddress(escapedShippingAddress);  }  DiscountDAO discountDAO = **new** DiscountDAO();  List<Discount> discounts = discountDAO.getActiveDiscounts();  CategorieDAO categorieDAO = **new** CategorieDAO(connection);  List<Categorie> categories = categorieDAO.getAllCategories();  **for** (Categorie p : categories) {  String escapedName = StringEscapeUtils.*escapeHtml4*(p.getName());  escapedName = StringEscapeUtils.*escapeEcmaScript*(escapedName);  p.setName(escapedName);    String escapedDecription = StringEscapeUtils.*escapeHtml4*(p.getDescription());  escapedDecription = StringEscapeUtils.*escapeEcmaScript*(escapedDecription);  p.setDescription(escapedDecription);  }  **if** ("admin".equals(page)) {  request.setAttribute("products", products);  request.setAttribute("orders", orders);  request.setAttribute("discounts", discounts);  request.setAttribute("categories", categories);  request.getRequestDispatcher("/views/admin.jsp").forward(request, response);  } **else** {  request.setAttribute("products", products);  request.setAttribute("categories", categories);  request.getRequestDispatcher("/views/product.jsp").forward(request, response);  } |

##### 6.1.8.4.3. Không bao giờ chèn dữ liệu người dùng trực tiếp vào mã HTML/JS

Không bao giờ chèn request.getParameter(...) trực tiếp vào trang. Luôn qua hàm mã hóa hoặc thẻ an toàn như <c:out>.

Chỉnh sửa tại trang product.jsp như sau:

|  |
| --- |
|  |

Hình 6.38. File product.jsp trước và sau khi được sửa

##### 6.1.8.4.4. Sử dụng thư viện bảo mật

Dùng các thư viện hỗ trợ mã hóa như:

StringEscapeUtils (Apache Commons Text)

|  |
| --- |
| <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-text</artifactId>  <version>1.10.0</version>  </dependency> |
| **import** org.apache.commons.text.StringEscapeUtils; // Thêm thư viện vào các class tương ứng |

OWASP Java Encoder Project

|  |
| --- |
| <dependency>  <groupId>org.owasp.encoder</groupId>  <artifactId>encoder</artifactId>  <version>1.2.3</version>  </dependency> |
| **import** org.owasp.encoder.Encode; // Thêm thư viện tương ứng  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  **int** productID = Integer.*parseInt*(request.getParameter("productID"));  **try** (Connection connection = ConnectionUtil.*DB*()) {  ProductDAO productDao = **new** ProductDAO(connection);  Product product = productDao.getProductById(productID);  String safeDescription = Encode.*forHtml*(product.getDescription());  product.setDescription(safeDescription);    request.setAttribute("product", product);  request.getRequestDispatcher("/views/product\_detail.jsp").forward(request, response);    } **catch** (Exception e) {  e.printStackTrace();  response.sendError(HttpServletResponse.***SC\_INTERNAL\_SERVER\_ERROR***, "Đã xảy ra lỗi khi kết nối với cơ sở dữ liệu.");  }  } |

Sử dụng framework có cơ chế chống XSS mặc định như Spring, JSF, hoặc JSP với JSTL.

##### 6.1.8.4.5. Kiểm thử lại với công cụ OWASP ZAP

|  |
| --- |
|  |

Hình 6.39. Lỗi Cross Site Script (XSS) đã được khắc phục

Sau khi đã fix các lỗ hổng và quét lại bằng công cụ OWASP ZAP với trang đã được chỉnh sửa, ta thấy các lỗ hổng về XSS đã không xuất hiện. Như vậy, chúng ta đã fix thành công các lỗ hổng về XSS bằng Input Validation, Contextual Output Encoding, Content Security Policy…

|  |
| --- |
|  |

Hình 6.40. Tấn công XSS thử lại

## 6.2. Kiểm tra thủ công

### 6.2.1. Brute Force Attack

#### 6.2.1.1. Khái niệm

**Brute Force Attack (tấn công vét cạn)** là một kỹ thuật tấn công phổ biến trong đó kẻ tấn công tự động thử nhiều tổ hợp giá trị đầu vào—chẳng hạn như tên người dùng, mật khẩu hoặc mã xác thực—cho đến khi tìm ra một tổ hợp hợp lệ. Các cuộc tấn công dạng này thường khai thác điểm yếu trong các cơ chế xác thực, đặc biệt khi ứng dụng không giới hạn số lần thử hoặc không có các biện pháp phòng ngừa hiệu quả

Tấn công brute-force là một phương pháp phổ biến mà hacker sử dụng để đoán tên người dùng và mật khẩu, cố gắng truy cập trái phép vào hệ thống. Hacker thường tự động hóa quá trình này bằng cách sử dụng danh sách các mật khẩu thông dụng, được gọi là "dictionary", có thể chứa hàng triệu mật khẩu khác nhau. Các công cụ chuyên dụng cho phép hacker thực hiện đăng nhập tự động nhiều lần với tốc độ cao.

Độ mạnh của mật khẩu phụ thuộc vào sự phức tạp của nó. Mật khẩu chỉ sử dụng chữ cái thường và không có ký tự đặc biệt hoặc chữ số có thể bị crack trong khoảng 2-10 phút. Tuy nhiên, mật khẩu kết hợp cả chữ hoa và chữ thường, cùng với một số chữ số, sẽ mất hàng chục năm để bị crack.

#### 6.2.1.2. Tấn công

Để tiến hành tấn công brute force đối với project, ta cần biết được thông tin username để có thể tấn công dò mật khẩu.

|  |
| --- |
|  |

Hình 6.41. Tìm kiếm username tồn tại

Ở đây, khi chúng ta tiến hành đăng ký tài khoản, nếu như username đã tồn tại thì hệ thống sẽ cho chúng ta một thông báo như trên hình, đây là một dấu hiệu cho ta biết được thông tin về username của người dùng

Tiếp theo, ta sẽ bắt gói tin khi đăng ký, và tiến hành dò tên tài khoản của một user nào đó đã tồn tại trong hệ thống.

|  |
| --- |
|  |

Hình 6.42. Bắt gói tin đăng ký bằng ZAP

Ở trên là nội dung của gói tin, ta tiến hành dùng ZAP Fuzzer để dò username đã tồn tại:

|  |
| --- |
|  |

Hình 6.43. dùng ZAP Fuzzer để dò username tồn tại

|  |
| --- |
|  |

Hình 6.44. dùng ZAP Fuzzer để dò username

Ta thêm payload cho phần username

|  |
| --- |
|  |

Hình 6.45. Add Payload để dò username

File được đưa vào là một file chứa tên cái username phổ biến được cung cấp bởi: <https://github.com/danielmiessler/SecLists/blob/master/Usernames/xato-net-10-million-usernames.txt>

Tiến hành Brute Force username:

|  |
| --- |
|  |

Hình 6.46. Brute Force username

Vậy thì làm thế nào ta biết được một task khi tiến hành Brute Force là đã dò ra được tài khoản đã tồn tại? Ta sẽ tiến hành xem kích thước trả về của response body để xem xét rằng gói tin nào nhận được một thông báo tài khoản đã tồn tại

Dựa vào sai khác kích thước của các gói tin trả về, ta quét được một gói tin như sau

|  |
| --- |
|  |

Hình 6.47. Dùng ZAP quét gói tin

Và đây chính xác là thông tin username đã tồn tại

Tiếp theo đây, ta sẽ tiến hành Brute Force mật khẩu của user này dựa trên một file chứa các mật khẩu thông dụng theo đường dẫn sau <https://github.com/duyet/bruteforce-database/blob/master/1000000-password-seclists.txt>

Tương tự như bước cài đặt Brute Force cho username, ta tiến hành Brute Force cho password

|  |
| --- |
|  |

Hình 6.48. Add Payload để tiến hành Brute Force

Và ta tiến hành Brute Force password:

|  |
| --- |
|  |

Hình 6.49. Brute Force password

Ta có thể thấy được là mật khẩu và tài khoản đã bị dò ra hoàn toàn

#### 6.2.1.3. Nguyên nhân

Trong thực tế, người dùng có thể bị lộ thông tin tên đăng nhập (username) qua nhiều kênh khác nhau nếu không được bảo vệ đúng cách. Một số nguyên nhân phổ biến bao gồm:

* Hệ thống phản hồi thông báo lỗi khác nhau cho username đúng và sai, từ đó cho phép attacker thực hiện user enumeration.
* Các chức năng như đăng ký, khôi phục mật khẩu hoặc biểu mẫu liên hệ vô tình tiết lộ sự tồn tại của tài khoản.
* Username bị rò rỉ từ các kênh không chính thống như log server, URL, mã JavaScript phía client, hoặc giao diện hiển thị công khai (bình luận, bài viết, v.v.).
* Người dùng sử dụng lại username trên nhiều nền tảng khác nhau, làm tăng khả năng thông tin bị lộ thông qua các vụ rò rỉ dữ liệu công khai.

Tuy nhiên, nguyên nhân cốt lõi khiến Brute Force Attack trở nên khả thi nằm ở việc người dùng sử dụng mật khẩu yếu hoặc phổ biến, chẳng hạn như 123456789, password, admin123, v.v. Những mật khẩu này có thể dễ dàng bị đoán ra chỉ sau vài lần thử, đặc biệt khi hệ thống không có cơ chế giới hạn số lần đăng nhập, không sử dụng CAPTCHA hoặc không áp dụng xác thực hai yếu tố (2FA).

Bên cạnh đó, nếu hệ thống không áp dụng các biện pháp phòng ngừa như rate limiting, tạm khóa tài khoản sau nhiều lần thất bại, hoặc ghi log cảnh báo hành vi bất thường, thì kẻ tấn công hoàn toàn có thể sử dụng công cụ tự động để thử hàng ngàn kết hợp username/password cho đến khi tìm được tài khoản hợp lệ.

#### 6.2.1.4. Khắc phục

##### 6.2.1.4.1 Góc độ nhà phát triển

Ở phía nhà phát triển, ta sẽ tiến hành thêm một phần mã xác thực vào form đăng nhập và việc này sẽ giúp chặn hoàn toàn các công cụ tự động brute force

Ta sẽ tạo một class chuyên dùng cho việc tạo mã xác thực AuthCodeUtil.java

|  |
| --- |
| **public** **class** AuthCodeUtil {  **public** **static** String generateCode(**int** length) {  String chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";  StringBuilder sb = **new** StringBuilder();  Random rnd = **new** Random();  **for** (**int** i = 0; i < length; i++) {  sb.append(chars.charAt(rnd.nextInt(chars.length())));  }  **return** sb.toString();  }  **public** **static** **void** refreshVerificationCode(HttpSession session) {  session.setAttribute("authCode", *generateCode*(6));  }  **public** **static** **boolean** isVerificationCodeValid(HttpServletRequest request) {  String userCode = request.getParameter("authCodeInput");  String sessionCode = (String) request.getSession().getAttribute("authCode");  **return** sessionCode != **null** && userCode != **null** && sessionCode.equalsIgnoreCase(userCode);  }  } |

Ở file Login\_Servlet.java, ta sẽ gọi hàm của Class AuthCodeUtil

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  AuthCodeUtil.*refreshVerificationCode*(request.getSession());  request.getRequestDispatcher("/views/login.jsp").forward(request, response);  } |

Khi đó ở login.jsp, ta sẽ sửa form đăng nhập lại, lúc này ta sẽ lấy mã xác thực đã được sinh ra

|  |
| --- |
| <%  String code = (String) session.getAttribute("authCode");  %>  <form id=*"auth-form"* action=*"/NenthomWeb/servlets/Login\_Servlet"* method=*"post"*>  <!-- Username -->  <div class=*"form-group"*>  <label for=*"username"*>Username</label>  <input type=*"text"* id=*"username"* name=*"username"* required>  </div>  <!-- Password -->  <div class=*"form-group"*>  <label for=*"password"*>Password</label>  <input type=*"password"* id=*"password"* name=*"password"* required>  </div>  <!-- Mã xác thực -->  <div class=*"form-group"*>  <label>Mã xác thực: <strong style="font-size: *18px*;"><%= code %></strong></label>  <input type=*"text"* name=*"authCodeInput"* required placeholder=*"Nhập mã xác thực"*>  </div>  <button type=*"submit"* class=*"login-page-button"*>Login</button>  </form> |

Giao diện hiển thị

|  |
| --- |
|  |

Hình 6.50. Giao diện hiển thị login

Khi người dùng bấm vào nút Login thì Login\_Servlet sẽ tiến hành kiểm tra mã xác thực ở doPost:

|  |
| --- |
| **if** (!AuthCodeUtil.*isVerificationCodeValid*(request)) {  AuthCodeUtil.*refreshVerificationCode*(request.getSession());  request.setAttribute("message", "Mã xác thực không đúng!");  request.setAttribute("error", **true**);  request.getRequestDispatcher("/views/login.jsp").forward(request, response);  **return**;  } |

Tương tự với phần đăng nhập, ở phần đăng ký ta cũng sẽ thêm mã xác thực:

Register\_Servlet

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  AuthCodeUtil.*refreshVerificationCode*(request.getSession());  request.getRequestDispatcher("/views/register.jsp").forward(request, response);  } |

register.jsp

|  |
| --- |
| <%  String code = (String) session.getAttribute("authCode");  %>  <form id=*"auth-form"* action=*"../servlets/Register\_Servlet"*  method=*"post"*>  <div class=*"form-group"*>  <label for=*"username"*>Username</label> <input type=*"text"*  id=*"username"* name=*"username"* placeholder=*"Enter your username"*  required>  </div>  <div class=*"form-group"*>  <label for=*"password"*>Password</label> <input type=*"password"*  id=*"password"* name=*"password"* placeholder=*"Enter your password"*  required>  </div>  <div class=*"form-group"*>  <label for=*"confirm-password"*>Confirm Password</label> <input  type=*"password"* id=*"confirm-password"* name=*"confirm-password"*  placeholder=*"Confirm your password"* required>  </div>  <div class=*"form-group"*>  <label>Mã xác thực: <strong style="font-size: *18px*;"><%= code %></strong></label>  <input type=*"text"* name=*"authCodeInput"* required  placeholder=*"Nhập mã xác thực"*>  </div>  <button type=*"submit"* id=*"submit-button"* class=*"login-page-button"*>SignUp</button>  </form> |

Register\_Servlet hàm doPost:

|  |
| --- |
| **if** (!AuthCodeUtil.*isVerificationCodeValid*(request)) {  AuthCodeUtil.*refreshVerificationCode*(request.getSession());  request.setAttribute("message", "Mã xác thực không đúng!");  request.setAttribute("error", **true**);  request.getRequestDispatcher("/views/register.jsp").forward(request, response);  **return**;  } |

Ngoài việc thêm mã xác thực, ta còn cần ràng buộc các điều kiện để tránh người dùng đặt một mật khẩu yếu

Bổ sung AuthCodeUtil.java

|  |
| --- |
| **public** **static** **boolean** isStrongPassword(String password) {  **if** (password == **null**) **return** **false**;  // Mật khẩu mạnh: ít nhất 8 ký tự, có chữ hoa, chữ thường, số và ký tự đặc biệt  String pattern = "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)(?=.\*[@#$%^&+=!]).{8,}$";  **return** password.matches(pattern);  } |

Khi đó về giao diện của phần đăng ký bổ sung hàm kiểm tra mật khẩu và thông báo ở phía giao diện cho người dùng

|  |
| --- |
|  |

Hình 6.51. Bổ dung hàm kiểm tra mật khẩu form đăng ký

|  |
| --- |
| **function** validatePassword() {  **const** password = document.getElementById("password").value;  **const** message = document.getElementById("pw-msg");  **const** strongRegex = /^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@#$%^&+=!]).{8,}$/;  **if** (!strongRegex.test(password)) {  message.innerText = "Mật khẩu phải có ít nhất 8 ký tự, bao gồm chữ hoa, chữ thường, số và ký tự đặc biệt.";  } **else** {  message.innerText = "";  }  } |

|  |
| --- |
|  |

Hình 6.52. Báo lỗi khi nhập mật khẩu sai

Khi người dùng đăng ký một tài khoản không đảm bảo độ mạnh mật khẩu thì sẽ bị trả lại về trang đăng ký

|  |
| --- |
| **if** (!AuthCodeUtil.*isStrongPassword*(password)) {  AuthCodeUtil.*refreshVerificationCode*(request.getSession());  request.setAttribute("message", "Mật khẩu không đủ mạnh. Vui lòng sử dụng ít nhất 8 ký tự bao gồm chữ hoa, chữ thường, số và ký tự đặc biệt.");  request.setAttribute("error", **true**);  request.getRequestDispatcher("/views/register.jsp").forward(request, response);  **return**;  } |

##### 6.2.1.4.2. Góc độ người dùng:

Đối với người dùng, để giảm thiểu rủi ro, điều quan trọng trước hết là phải nâng cao nhận thức người dùng về bảo mật. Người dùng cần hiểu rằng việc đặt mật khẩu yếu không chỉ là một sơ suất nhỏ, mà là một mối đe dọa nghiêm trọng đối với toàn bộ hệ thống tài khoản cá nhân.

Người dùng nên có trách nhiệm hơn trong việc lựa chọn mật khẩu, ưu tiên sử dụng mật khẩu mạnh, duy nhất cho từng nền tảng, bao gồm tối thiểu 8 ký tự, kết hợp giữa chữ hoa, chữ thường, số và ký tự đặc biệt. Ngoài ra, cần thay đổi định kỳ và tuyệt đối không sử dụng lại mật khẩu cũ hoặc phổ biến.

|  |
| --- |
|  |

Hình 6.53. Giao diện web Strong Password

Để hỗ trợ người dùng đánh giá độ mạnh của mật khẩu, hiện nay có rất nhiều công cụ trực tuyến có thể đánh giá mức độ an toàn của mật khẩu mà người dùng đặt, chẳng hạn như:

https://www.strongpasswordgenerator.org/vi/password-strength-checker/.

Những công cụ này giúp người dùng hiểu rõ hơn về mức độ an toàn của mật khẩu mà họ đang sử dụng, từ đó nâng cao ý thức bảo vệ tài khoản.

### 6.2.2 Broken Access Control

#### 6.2.2.1. Khái niệm

Broken Access Control (kiểm soát truy cập không đúng cách) là một trong những lỗ hổng nghiêm trọng thường gặp trong các ứng dụng web hiện đại. Theo báo cáo OWASP Top 10 năm 2021, đây là lỗ hổng đứng đầu danh sách về mức độ phổ biến và ảnh hưởng. Lỗi này phát sinh khi hệ thống không áp dụng đầy đủ hoặc sai lệch các cơ chế kiểm soát quyền truy cập, từ đó cho phép người dùng vượt quyền – tức là truy cập vào dữ liệu hoặc chức năng không thuộc phạm vi được cấp phép. Hậu quả của lỗ hổng này có thể bao gồm việc truy xuất trái phép dữ liệu nhạy cảm, chỉnh sửa thông tin của người khác, hoặc thực thi các thao tác quản trị mà không có quyền hợp lệ.

#### 6.2.2.2. Tấn công

Lỗ hổng Broken Access Control cho phép người dùng không có quyền quản trị (vai trò customer) có thể truy cập trực tiếp vào trang giao diện quản trị (admin.jsp) thông qua việc nhập trực tiếp đường dẫn URL: <http://localhost:8080/NenthomWeb/views/admin.jsp>

Mặc dù người dùng không phải admin không nhận được đầy đủ nội dung trang quản trị (giao diện hiển thị trắng hoặc thiếu dữ liệu), nhưng các thành phần HTML như nút “Add Product”, “Add Discount”, v.v. vẫn được render trên trình duyệt, và hoàn toàn có thể kích hoạt tương ứng với các servlet xử lý như AddProductServlet, AddDiscountServlet.

|  |
| --- |
|  |

Hình 6.54. Giao diện admin khi truy cập không có quyền admin

Chi tiết về lỗi:

* Người dùng đăng nhập bằng tài khoản customer bình thường.
* Truy cập thủ công URL: <http://localhost:8080/NenthomWeb/views/admin.jsp>
* Dù không được hiển thị đầy đủ dữ liệu quản trị do không phải luồng đăng nhập chính thống (Login bằng account Admin), các nút chức năng vẫn có thể sử dụng.
* Khi nhấn vào các nút như "Thêm sản phẩm", hệ thống vẫn hiển thị ra giao diện để thực hiện thao tác, và khi người dùng nhập đủ thông tin về sản phẩm cũng như ưu đãi, v.v và khi ấn Button, hành động vẫn diễn ra tương ứng như mong muốn do các servlet không kiểm tra quyền hạn.

#### 6.2.2.3. Nguyên nhân

Trang admin.jsp không có cơ chế kiểm tra quyền truy cập dựa vào role trong session.

Các servlet như AddProductServlet, AddDiscountServlet cũng không thực hiện kiểm tra quyền hạn trước khi xử lý yêu cầu từ người dùng gửi đến.

Hệ thống phụ thuộc hoàn toàn vào luồng điều hướng chính thống (sau khi đăng nhập đúng vai trò) mà bỏ qua kiểm soát truy cập đối với truy cập trực tiếp qua URL.

Mức độ nghiêm trọng của lỗi:

* Cao: Cho phép người dùng không có quyền thực hiện hành động quản trị như thêm sản phẩm, chỉnh sửa mã giảm giá, truy cập thông tin nội bộ.
* Có thể dẫn đến mất dữ liệu, thao túng hệ thống, hoặc rò rỉ thông tin nhạy cảm.

#### 6.2.2.4. Khắc phục

Sau khi phát hiện lỗ hổng Broken Access Control trong hệ thống — cụ thể là người dùng có thể truy cập trực tiếp vào các trang như admin.jsp dù không có quyền, ta tiến hành các bước khắc phục như sau: tạo lớp Java lọc truy cập (Filter) bảo vệ trang quản trị tên AdminPageFilter.java được tạo tại package filters, có chức năng kiểm tra quyền truy cập của người dùng đến trang admin.jsp.

* URL áp dụng: /views/admin.jsp
* Công nghệ sử dụng: @WebFilter của Servlet API
* Hành vi:
* Nếu người dùng không thuộc vào role "manager" → chuyển hướng đến unauthorized.jsp
* Nếu người dùng thuộc vào role "manager" nhưng truy cập admin.jsp thủ công (không đúng flow hệ thống) → tự động chuyển hướng về servlet chính /servlets/DSProduct\_Servlet?page=admin để hiển thị nội dung hợp lệ.

|  |
| --- |
| @WebFilter("/views/admin.jsp")  public class AdminPageFilter implements Filter {  @Override  public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)  throws IOException, ServletException {  HttpServletRequest request = (HttpServletRequest) req;  HttpServletResponse response = (HttpServletResponse) res;  HttpSession session = request.getSession(false);  String role = (session != null) ? (String) session.getAttribute("role") : null;  if (!"manager".equals(role)) {  response.sendRedirect(request.getContextPath() + "/views/unauthorized.jsp");  return;  }  response.sendRedirect(request.getContextPath() + "/servlets/DSProduct\_Servlet?page=admin");  }  } |

Sau khi thêm đoạn code trên, tiền hành test lại bên phía customer: khi truy cập bất hợp pháp vào url <http://localhost:8080/NenthomWeb/views/admin.jsp> sẽ có một thông báo “Bạn không có quyền truy cập vào trang này”

|  |
| --- |
|  |

Hình 6.55. Giao diện admin thông báo lỗi không có quyền truy cập

### 6.2.3. Buffer Overflow

#### 6.2.3.1. Khái niệm

Trong lĩnh vực bảo mật máy tính và lập trình, Buffer Overflow (tràn bộ đệm) là một lỗi xảy ra khi dữ liệu vượt quá giới hạn của vùng bộ nhớ tạm (buffer) được cấp phát sẵn. Tình trạng này thường xảy ra khi chương trình cho phép người dùng nhập vào nhiều dữ liệu hơn so với kích thước tối đa của bộ đệm, khiến phần dư bị ghi đè lên các khu vực nhớ lân cận. Đây là một trong những lỗ hổng phổ biến và nguy hiểm, vì nó có thể dẫn đến hành vi không xác định như truy cập sai vùng nhớ, làm sập chương trình, hoặc tệ hơn, bị khai thác để thực thi mã độc từ xa — tạo điều kiện cho kẻ tấn công kiểm soát hệ thống.

#### 6.2.3.2. Tấn công

Chèn một lượng lớn khoảng 1000 kí tự vào ô input nhập tên trong mục quản lý thông tin cá nhân, ta nhận thấy khi nhấn cập nhật hệ thống sẽ bị vỡ và trên server log ra lỗi:

|  |
| --- |
|  |

Hình 6.56. Giao diện profile

|  |
| --- |
|  |

Hình 6.57. Giao diện profile thông báo lỗi cập nhật

Màn hình Console trong Eclipse

|  |
| --- |
|  |

Hình 6.58. Màn hình console trong eclipse thông báo lỗi

#### 6.2.3.3. Nguyên nhân

Do hệ thống không xác thực dữ liệu đầu vào và không có ràng buộc về dữ liệu nên khi nhập một lượng lớn kí tự thì không thể xử lý được dẫn đến hệ thống không thể phản hồi.

#### 6.2.3.4. Khắc phục

Giới hạn số lượng kí tự nhập vào ô input có lẽ là cách hữu hiệu trong trường hợp này vì vậy ở file user\_infor.jsp trong src\main\webapp\WEB-INF\views\profile.jsp ta thêm thuộc tính max-length:100 vào ô input nhập email

max-length: 15 vào ô input nhập số điện thoại

max-length: 255 vào ô input nhập địa chỉ

Điều này giúp giới hạn độ dài của tên người dùng được hiển thị trên giao diện người dùng và trong database cũng sẽ lưu lại được thông tin tránh buffer overflow.

|  |
| --- |
|  |

Hình 6.59. Code file profile.jsp

Tiếp đến cần sửa lại trong src/main/java/servlets/UpdateProfile\_Servlet.java để cắt chuỗi từ người dùng nhập vào. Bởi vì thiết lập maxlength ở phía client là chưa đủ. Kẻ tấn công có thể mở devtool để chỉnh sửa độ dài của maxlength nên server cũng cần phải được bảo vệ.

|  |
| --- |
|  |

Hình 6.60. Code file UpdateProfile\_Servlet

Ta nhập lại 1000 kí tự và kết quả hệ thống hiển thị tối đa 255 kí tự ở ô input địa chỉ và trong database thông tin đã được lưu.

|  |
| --- |
|  |

Hình 6.61. Tấn công Buffer Overflow ở trang profile

|  |
| --- |
|  |

Hình 6.62. Tấn công Buffer Overflow profile đã khắc phục

|  |
| --- |
|  |

Hình 6.63. Database bảng Users sau khi tấn công khắc phục Buffer Overflow

=> Tương tự với các ô input khác, cần phải giới hạn lại độ dài kí tự được nhập ở phía client trong các file admin.jsp

|  |
| --- |
|  |

Hình 6.64. Code admin.jsp tương tự

|  |
| --- |
|  |

Hình 6.65. Code AddProduct\_Servlet tương tự

### 6.2.4. Improper Input Validation

#### 6.2.4.1. Khái niệm

Improper Input Validation là lỗi xảy ra khi ứng dụng không có các phương thức để xác thực đầu vào hoặc có nhưng không đúng, dẫn đến việc đầu vào trở nên không đáng tin cậy và không được như ứng dụng mong đợi. Đầu vào của ứng dụng là bất kỳ dữ liệu nào được cung cấp từ nguồn bên ngoài, chẳng hạn dữ liệu từ form được người dùng nhập vào, chuỗi truy vấn, POST request, cookie, API response...

Khi đầu vào không được xác thực chính xác, kẻ tấn công có thể lợi dụng lỗ hổng này để khai thác thông tin từ ứng dụng thông qua việc thực hiện một số hình thức tấn công như SQL Injection, Cross-Site Scripting (XSS) và các kiểu tấn công khác

#### 6.2.4.2. Tấn công

Trong project, ở phần đăng ký tuy là đã được cài đặt các điều kiện giới hạn cho mật khẩu khi đã sửa lỗi (xem mục 2.1), tuy nhiên, ta vẫn có thể chèn các dữ liệu không hợp lệ khi đăng ký

|  |
| --- |
|  |

Hình 6.66. Giao diện đăng ký

Sau đó ta ZAP để bắt lấy gói tin và tiến hình sửa đổi tên username

|  |
| --- |
|  |

Hình 6.67. ZAP bắt gói tin đăng ký

Và gửi gói tin đi với username là rỗng, sau đó ta quay lại trang đăng nhập

|  |
| --- |
|  |

Hình 6.68. Quay lại giao diện đăng nhập

Vì ở phía giao diện, các ô input đều đã cài đặt require, nên ta sẽ ghi tạm vào một giá trị bất kỳ, sau đó dùng ZAP để sửa đổi

|  |
| --- |
|  |

Hình 6.69. Dùng ZAP để sửa đổi

Và kết quả là đã đăng nhập được

|  |
| --- |
|  |

Hình 6.70. Đăng nhập thành công giao diện trang chủ

#### 6.2.4.3. Nguyên nhân

Nguyên nhân chính là do không có các phương thức để kiểm tra đầu vào từ dữ liệu của người dùng nhập vào.

#### 6.2.4.4. Khắc phục

Ta có thể dùng các thư viện như Apache Common Validator, Hibernate Validator hoặc thậm chí là tự viết hàm kiểm tra đầu vào cho các dữ liệu được gửi về phía server

Ở project, ta sẽ sử dụng Hibernate Validator bằng cách thêm đoạn sau vào pom.xml

|  |
| --- |
|  |

Hình 6.71. Thêm thư viện vào file pom.xml

Bổ sung cho model TaiKhoan.java

|  |
| --- |
|  |

Hình 6.72. Bổ sung model TaiKhoan

Tại Register\_Servlet

|  |
| --- |
|  |

Hình 6.73. Code Register\_Servlet

Ta sửa phần hiển thị thông báo lỗi ở register.jsp

|  |
| --- |
|  |

Hình 6.74. Code register.jsp

Như vậy là ta đã ngăn chặn được việc các dữ liệu không hợp lệ được ghi vào database

|  |
| --- |
|  |

Hình 6.75. Giao diện ZAP bắt gói tin

|  |
| --- |
|  |

Hình 6.76. Thông báo lỗi

Ngoài ra, bổ sung thêm phần thông báo cho login.jsp và register.jsp về username

|  |
| --- |
|  |

Hình 6.77. Code bổ bung login.jsp

|  |
| --- |
|  |

Hình 6.78. Code thông báo lỗi cho login.jsp

### 6.2.5. Vulnerable and Outdated Components

#### 6.2.5.1. Khái niệm

Vulnerable and Outdated Components là tình trạng khi một ứng dụng hoặc hệ thống sử dụng các thành phần phần mềm (thư viện, framework, plugin, hoặc module) bị lỗi thời hoặc chứa lỗ hổng bảo mật đã được phát hiện. Những thành phần này có thể bị khai thác bởi kẻ tấn công để thực hiện các hành vi xâm nhập, đánh cắp dữ liệu, hoặc phá hoại hệ thống.

#### 6.2.5.2. Kiểm tra

* Bước 1: Tiến hành thêm plugin vào file pom.xml để tiến hành quét các thư viện xem có chứa lỗ hổng bảo mật hay không.

|  |
| --- |
|  |

Hình 6.79. Thêm plugin file pom.xml

* Bước 2: Tiến hành build lại dự án Maven trong eclipse

|  |
| --- |
|  |

Hình 6.80. Thêm plugin file pom.xml

* Kết quả sau khi build xong sẽ có 1 file dependency-check-report.html trong thư mục target của dự án.

|  |
| --- |
|  |

Hình 6.81. File dependency-check-report.html

* Mở file .html lên để xem chi tiết về lỗ hổng đã được quét.

|  |
| --- |
|  |

Hình 6.82. Mở file dependency-check-report.html

* Giải thích một số thuật ngữ:
* CPE (Common Platform Enumeration): định danh phần mềm liên quan đến lỗ hổng.
* Confidence: độ tin cậy khi mapping thư viện → lỗ hổng (thấp có thể là false positive).
* CVE Count: số lỗ hổng tìm được với thư viện đó.
* Severity: mức độ nghiêm trọng (Low / Medium / High / Critical).

#### 6.2.5.3. Nguyên nhân

Các thư viện bên thứ ba (third-party) bạn dùng có CVE (Common Vulnerabilities and Exposures) đã được công bố:

Các lỗ hổng này có thể do:

* Sử dụng phiên bản thư viện đã lỗi thời.
* Các lỗ hổng chưa được vá trong bản hiện tại.
* Mức confidence cao (Highest) → chứng minh lỗ hổng tồn tại chắc chắn.

#### 6.2.5.4. Khắc phục

Sau khi kiểm tra các dependency đó. Chúng ta sẽ suppress những lỗ hổng được cho là false positive hoặc không ảnh hưởng đến bạn (do không dùng đến tính năng đó).

* Bước 1: Tạo file suppression.xml. Mục đích để dependency-check-owasp bỏ qua các: Cảnh báo không chính xác (còn gọi là "false positive") mà công cụ đưa ra.

Tức là, công cụ "nghĩ" rằng một thư viện nào đó có vấn đề bảo mật, nhưng sau khi bạn kiểm tra, bạn biết chắc rằng:

* Thư viện đó thực sự an toàn.
* Cảnh báo đó không liên quan gì đến thư viện của bạn (ví dụ: công cụ nhận diện nhầm tên).
* Phiên bản bạn đang dùng không bị ảnh hưởng bởi lỗ hổng được nhắc đến.

|  |
| --- |
|  |

Hình 6.83. File suppression.xml

Với các suppress được cung cấp từ dependency-check của oswap

|  |
| --- |
|  |

Hình 6.84. Các suppress cung cấp từ dependency-check

* Bước 2: Import file đó vô plugin dependency-check của owasp.

|  |
| --- |
|  |

Hình 6.85. Import file vào plugin

* Bước 3: Tiến hành build lại dự án và mở file .html sau khi build thành công.

|  |
| --- |
|  |

Hình 6.86. File .html sau khi build thành công

=> Đã không còn lỗ hổng ở các dependency trong dự án.

### 6.2.6. Insecure Client-Side Validation / Parameter Tampering

#### 6.2.6.1. Khái niệm

Insecure Client-Side Validation là một lỗ hổng bảo mật xảy ra khi ứng dụng chỉ thực hiện kiểm tra dữ liệu đầu vào (validation) trên phía client (trình duyệt), mà không kiểm tra lại trên server. Điều này khiến kẻ tấn công dễ dàng chỉnh sửa, giả mạo hoặc bỏ qua các kiểm tra chỉ bằng cách can thiệp vào HTML/JavaScript hoặc sử dụng công cụ như Burp Suite, OWASP ZAP, Fiddler,...

Parameter Tampering (giả mạo tham số) là một kỹ thuật tấn công mà kẻ tấn công thay đổi các tham số được gửi từ client đến server để đạt được các hành vi không mong muốn như:

* Truy cập trái phép
* Thay đổi giá tiền
* Nâng quyền
* Thực hiện các hành động bị giới hạn

#### 6.2.6.2. Tấn công

* Bước 1: Tiến hành vô checkout giỏ hàng

|  |
| --- |
|  |

* Bước 2: Mở devtool -> Sửa lại giá trị totalAmount đã được ẩn đi ở client.

|  |
| --- |
|  |

* Bước 3: Tiến hành ấn nút Thanh toán để checkout.

|  |
| --- |
|  |
|  |

=> Kết quả: Đã thành công chỉnh sửa lại tiền hóa đơn.

#### 6.2.6.3. Nguyên nhân

* Do ở phía server không xác thực lại thông tin được truyền phía Client.

|  |
| --- |
|  |

Điều này cho phép kẻ tấn công giả mạo hoặc thay đổi dữ liệu như voucherId, totalAmount, hoặc các thông tin quan trọng khác trong quá trình gửi yêu cầu (request). Nếu server tin tưởng hoàn toàn vào dữ liệu do client gửi lên mà không kiểm tra lại với cơ sở dữ liệu hoặc logic nghiệp vụ, thì kẻ tấn công có thể: Thay đổi tổng tiền thanh toán.

#### 6.2.6.4. Khắc phục

* Xác thực dữ liệu phía server: Luôn kiểm tra và xác minh lại toàn bộ thông tin quan trọng (voucher, user, giỏ hàng, tổng tiền,…) ở phía backend.
* Bước 1: Tiến hành tính toán lại giá tiền trong giỏ hàng ở server. Trong file src/main/java/servlets/CheckOut\_Servlet.java

|  |
| --- |
|  |

* Bước 2: Kiểm tra lại voucher ID có tồn tại hay không? Nếu có sử dụng, không thì thông báo lỗi.

|  |
| --- |
|  |
|  |

* Tiến hành tấn công lại. Chỉnh totalAmount = 0. Tiền thực là $17.09 với voucher sử dụng được.

|  |
| --- |
|  |
|  |
|  |

=> Kết quả phòng chống thành công.

* Thử với trường hợp voucher không tồn tại.

|  |
| --- |
|  |
|  |

=> Kết quả: Thông báo lỗi và không cho thanh toán.

## 6.3. Các lỗi không phát hiện

### 6.3.1. Insecure Direct Object Reference (IDOR)

#### 6.3.1.1. Khái niệm

Insecure Direct Object Reference (IDOR) xảy ra khi ứng dụng cho phép truy cập trực tiếp đến các tài nguyên nội bộ như tệp tin, cơ sở dữ liệu hoặc đối tượng hệ thống thông qua các tham số do người dùng cung cấp, mà không thực hiện kiểm tra đầy đủ về quyền truy cập.

Lỗ hổng này tạo điều kiện để kẻ tấn công thao túng các tham số (chẳng hạn như ID người dùng, tên tệp, mã đơn hàng...) nhằm truy xuất đến các đối tượng không thuộc quyền sở hữu của họ để tiến hành những thao tác tấn công như khai thác, chỉnh sửa hoặc thậm chí là xóa bỏ các thông tin nhạy cảm. Nguy cơ phát sinh từ việc ứng dụng sử dụng giá trị người dùng nhập vào để tham chiếu trực tiếp đến tài nguyên nội bộ mà không kiểm tra xem người dùng có được phép truy cập tài nguyên đó hay không.

Và vì vậy, để ngăn chặn được lỗi này, ta cần thiết lập việc kiểm tra quyền truy cập đến các tài nguyên có trong hệ thống thông qua một cơ chế xác thực nhằm kiểm soát toàn bộ mọi hoạt động truy cập từ phía người dùng.

#### 6.3.1.2. Xác định dấu hiệu

Để tiến hành kiểm tra lỗi IDOR, đầu tiên ta cần xem xét qua các tính năng mà người dùng có thể tương tác đối với hệ thống này và gói tin tương ứng:

* Xem thông tin cá nhân

|  |
| --- |
|  |

* Chỉnh sửa thông tin cá nhân

|  |
| --- |
|  |

* Xem chi tiết sản phẩm

|  |
| --- |
|  |

* Thêm sản phẩm vào giỏ hàng

|  |
| --- |
|  |

* Xem chi tiết giỏ hàng

|  |
| --- |
|  |

* Xem thông tin chi tiết đơn hàng

|  |
| --- |
|  |

* Thanh toán đơn hàng

|  |
| --- |
|  |

Dựa vào khái niệm, dấu hiệu của lỗ hổng IDOR là việc hệ thống sử dụng thông tin phía người dùng để thực hiện truy xuất dữ liệu. Khi đó, ta có thể thu gọn phạm vi kiểm tra lại, chỉ tiến hành kiểm tra đối với những tính năng mà gói tin có lấy thông tin từ phía người dùng để thực hiện truy xuất dữ liệu (qua các tham số trong phần header/body và là các tính năng truy xuất thông tin cá nhân, giỏ hàng, …), ta được:

* Chỉnh sửa thông tin cá nhân (có tham số trong body)
* Xem chi tiết sản phẩm (có tham số trong URL)
* Thêm sản phẩm vào giỏ hàng (có tham số trong body)
* Xem thông tin chi tiết đơn hàng (có tham số trong URL)
* Thanh toán đơn hàng (có tham số trong body)

Tiếp theo, ta sẽ thu gọn phạm vi kiểm tra lại hơn nữa.

Chúng ta sẽ bỏ qua chức năng xem chi tiết sản phẩm, vì đây là chức năng mà bất cứ user nào cũng có thể sử dụng được, không vi phạm nguyên tắc về tính xác thực người dùng, đây là thông tin được công khai cho toàn bộ user.

Chúng ta sẽ bỏ qua chức năng thêm sản phẩm vào giỏ hàng. Vì tham số product Id là thông tin công khai và hợp lệ để người dùng gửi lên, vì sản phẩm là đối tượng được chia sẻ công khai trong hệ thống, không bị ràng buộc quyền sở hữu. Khác với các đối tượng riêng tư như thông tin cá nhân, đơn hàng hay giỏ hàng người dùng, thì productID chỉ đóng vai trò tham chiếu đến một tài nguyên công khai (sản phẩm), không phải là thông tin nhạy cảm. Và cũng vì thế nên quantity cũng chỉ giữ đúng vai trò của mình là truyền đi số lượng sản phẩm, không dùng để khai thác IDOR.

Chúng ta sẽ bỏ qua chức năng thanh toán đơn hàng. Mặc dù có truyền thông tin nhạy cảm - tham số address (địa chỉ), nhưng request không chứa bất kỳ mã định danh (ID) nào liên quan đến đơn hàng, người dùng khác hoặc các đối tượng cần phân quyền. Do đó không có cơ chế nào để người dùng có thể can thiệp vào đơn hàng của người khác, tức là lỗi IDOR sẽ không xuất hiện ở đây

Chúng ta sẽ bỏ qua chức năng xem thông tin chi tiết đơn hàng. Mặc dù có tham số discountID trong URL, người dùng có thể thay đổi URL này để thay đổi mã giảm giá mà mình đã áp dụng vào đơn hàng, tuy nhiên về mặt thiết kế thì mã giảm giá được áp dụng cho toàn bộ khách hàng, đây cũng là một loại thông tin công khai. Vì vậy lỗi IDOR sẽ không xảy ra

Vậy ta cần kiểm tra:

* Chỉnh sửa thông tin cá nhân (có tham số trong body)

#### 6.3.1.3. Tiến hành kiểm tra

Chỉnh sửa thông tin cá nhân

|  |
| --- |
|  |

Hình trên đây là gói tin thực hiện thao tác chỉnh sửa thông tin cá nhân lấy được từ giao diện người dùng. Ta có thể thấy, ở đây có tiến hành sử dụng userId ở phía người dùng truyền sang để tiến hành thực hiện thao tác. Ta sẽ kiểm tra xem phần userId này được dùng như thế nào ở trong project, cụ thể là ở UpdateProfile\_Servlet

|  |
| --- |
|  |

Khi phân tích mã nguồn backend ở trên, ta thấy phần giá trị userId được gửi từ client không hề sử dụng. Thay vào đó, ứng dụng luôn lấy userID từ session của người dùng hiện tại

Vậy nên ta có thể kết luận rằng chức năng này không bị lỗ hổng IDOR

### 6.3.2. SQL Injection

#### 6.3.2.1. Khái niệm

SQL Injection là một kỹ thuật xâm nhập vào hệ thống thông qua việc chèn các đoạn mã SQL không hợp lệ vào các trường nhập liệu của ứng dụng. Thay vì cung cấp dữ liệu đúng đắn, kẻ tấn công tận dụng lỗ hổng này để thêm và thực thi các truy vấn SQL không đúng mục đích, từ đó truy cập và khai thác dữ liệu trong cơ sở dữ liệu.

Thay vì cung cấp dữ liệu hợp lệ, SQL Injection cho phép kẻ tấn công chèn và thực thi các câu lệnh SQL không hợp lệ để truy xuất thông tin người dùng từ cơ sở dữ liệu. Điều này có thể dẫn đến các hậu quả như:

* Hack tài khoản cá nhân.
* Ẩn danh và sao chép dữ liệu trang web hoặc hệ thống.
* Thay đổi thông tin nhạy cảm trong hệ thống.
* Xóa hoặc phá hủy dữ liệu quan trọng.
* Người dùng có thể truy cập vào ứng dụng với đặc quyền không phù hợp, kể cả là quản trị viên.

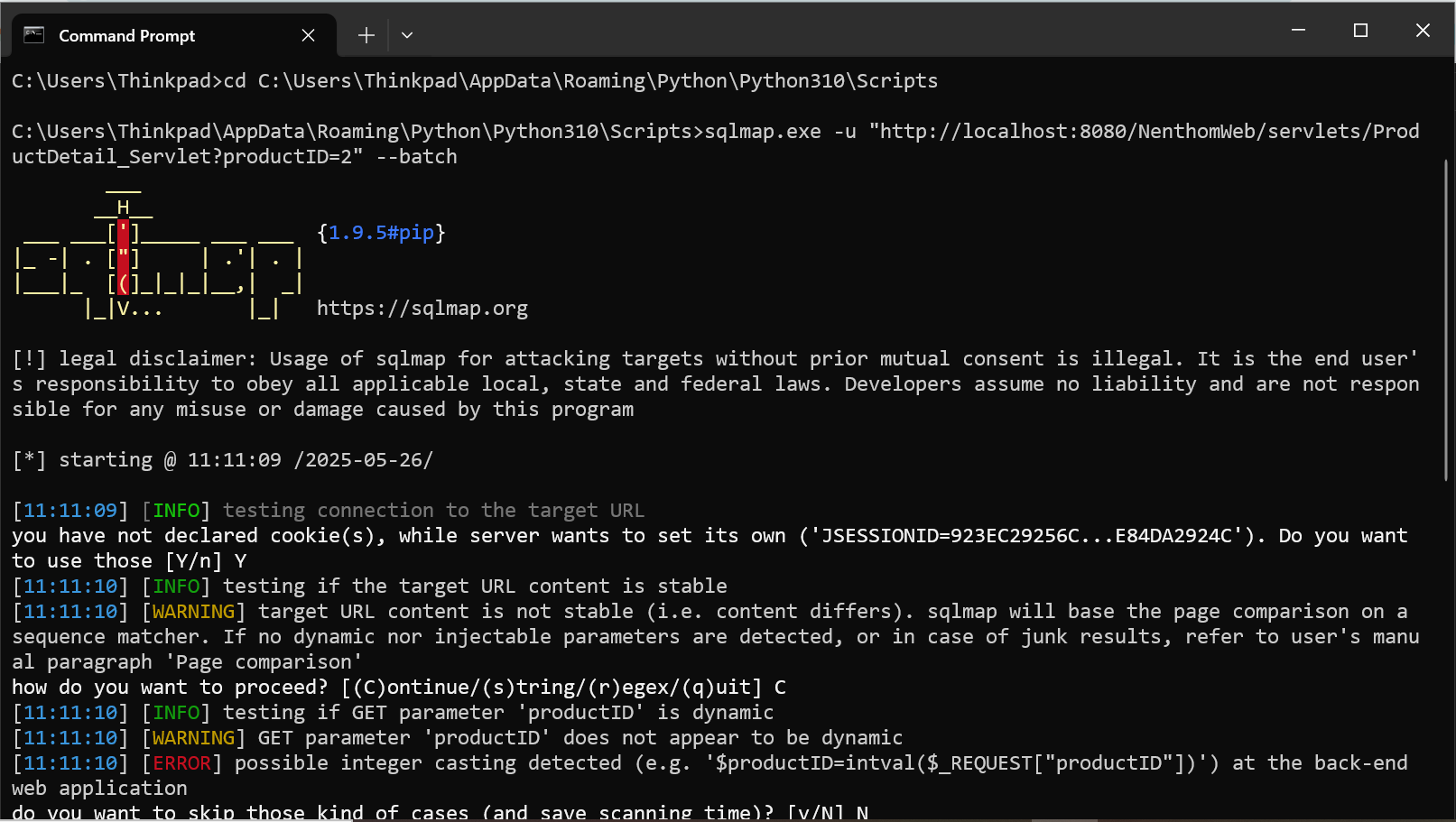
#### 6.3.2.2. Tấn công

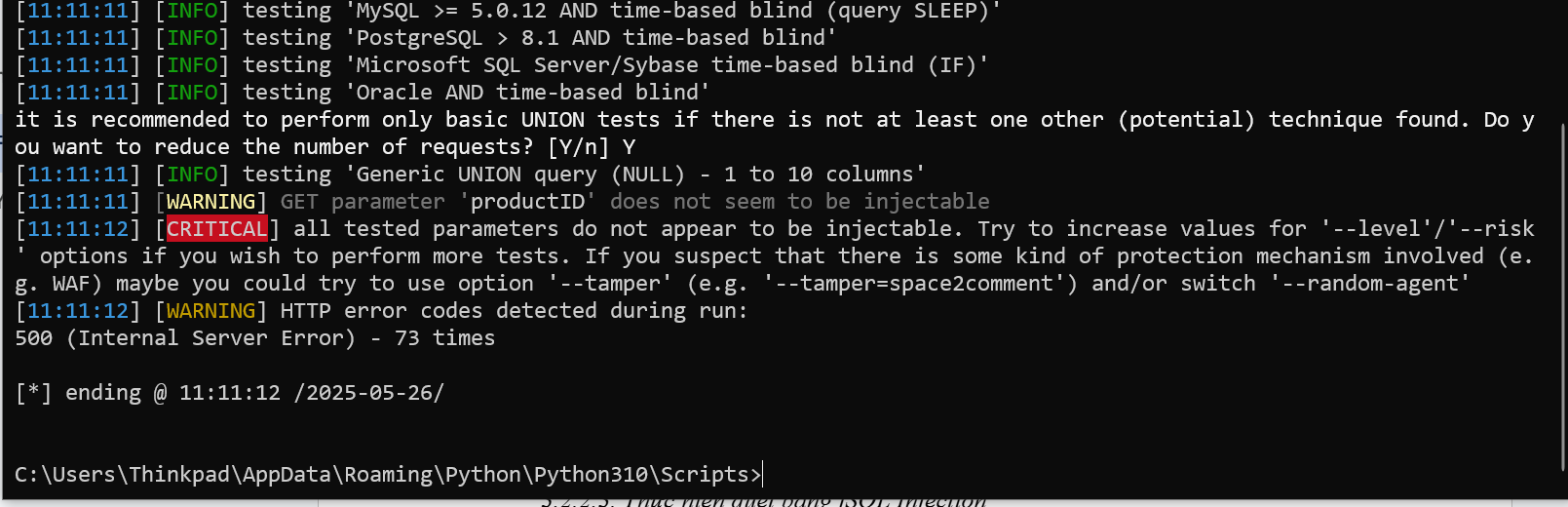
##### 6.3.2.2.1. Thực hiện quét bằng ZAP

|  |
| --- |
|  |

Sau khi tiến hành quét lỗ hổng bằng OWASP ZAP ta thấy ZAP phát hiện lỗ hổng bảo mật về lỗi SQL Injection nhưng đây chỉ là lỗ hổng mà ZAP nghi ngờ có thể tấn công “SQL injection may be possible” nên chưa thể chắc chắn rằng có thật sự bị SQL injection hay không nhưng do đây là các lỗ hổng thường xuyên bị các attacker nhắm đến và tấn công nên chúng ta sẽ thực hiện quét bằng các công cụ khác để kiểm tra.

##### 6.3.2.2.2. Thực hiện quét bằng SQLMAP





Trong quá trình kiểm thử tham số productID của URL:

|  |
| --- |
| <http://localhost:8080/NenthomWeb/servlets/ProductDetail_Servlet?productID=2> |

Sử dụng công cụ SQLMap, kết quả phân tích cho thấy:

[CRITICAL] Không phát hiện được tham số nào có khả năng injectable (chèn được mã SQL).

SQLMap thử nhiều kỹ thuật kiểm tra như:

* MySQL/PostgreSQL/MSSQL/Oracle time-based blind, UNION-based, Generic UNION query nhưng không thành công.
* Xuất hiện nhiều lỗi HTTP 500 (Internal Server Error), có thể do hệ thống xử lý lỗi nội bộ hoặc áp dụng các biện pháp bảo vệ (WAF, kiểm tra đầu vào...).

Kết luận: Tại thời điểm kiểm tra, không phát hiện được lỗ hổng SQL Injection trên endpoint đã chỉ định. Hệ thống có thể đã có cơ chế kiểm soát đầu vào hoặc xử lý lỗi tốt, tuy nhiên cần tiếp tục kiểm thử bằng các công cụ khác.

Vậy tại sao ZAP lại bắt được nhưng SQLMAP lại không, vì ZAP thường đánh dấu là "possible SQL injection", nghĩa là chỉ nghi ngờ dựa trên phản hồi HTTP khác nhau khi thử payload như ' or 1=1 --, chứ không chắc chắn 100%.

Trong khi đó, sqlmap sẽ thực sự thử khai thác SQLi, nên nếu nó không xác định được tham số động và có thể tiêm, thì nó báo là "not injectable".

→ ZAP = phát hiện dựa trên dấu hiệu.

→ sqlmap = thực sự khai thác.

##### 6.3.2.2.3. Thực hiện quét bằng jSQL Injection

|  |
| --- |
|  |

Công cụ JSQL nhận định tham số productID có dấu hiệu dễ bị tấn công kiểu Blind SQL Injection với AND, tuy nhiên:

* Khi thực hiện lấy dữ liệu (Fetching metadata), hệ thống trả về kết quả không hợp lệ (invalid/incomplete).
* Phản hồi của server chứa ký tự lạ, gợi ý việc kiểm tra bị cản trở hoặc mã hóa.
* Xuất hiện thông báo Boolean false positive, nghĩa là: phát hiện có thể là dương tính giả.

Kết luận tại thời điểm kiểm tra với JSQL:

* Chưa thể khẳng định chắc chắn có tồn tại lỗ hổng SQL Injection.
* Công cụ nhận định có khả năng tồn tại lỗ hổng kiểu Blind SQL Injection, nhưng phản hồi bất thường từ server khiến việc trích xuất dữ liệu không thành công.
* Khả năng cao là hệ thống có cơ chế kiểm tra đầu vào hoặc bảo vệ tạm thời, dẫn đến kết quả nghi ngờ là false positive.

##### 6.3.2.2.4. Thực hiện quét thủ công bằng tay

Để thực hiện tấn công thủ công SQLi, chúng ta có thể tham khảo và sử dụng các cheat sheet từ OWASP hoặc Github SQL Injection Payload List (<https://github.com/payloadbox/sql-injection-payload-list>)

Dưới đây là một số payload được sử dụng để tấn công SQLi trong project:

|  |
| --- |
| ' AND 1=2 UNION SELECT " ' AND 1=2 UNION SELECT @@version -- -" -- -  ' OR '1' = '1  AND 1=1  UNION SELECT 1, version() limit 1,1  ORDER BY 10–  UNION SELECT 1,null,null–  ' AND 1 = 2 UNION SELECT "PAYLOAD" -- -  ' OR ASCII(SUBSTRING(username,1,1))=97 AND '1'='1  UNION ALL SELECT 1,2,3,4,5,6,7,8,9,10  UNION SELECT @@VERSION,SLEEP(5),USER(),4 |

Lần lượt thực hiện tấn công với các payload trên:

|  |
| --- |
| http://localhost:8080/NenthomWeb/servlets/ProductDetail\_Servlet?productID=1' AND 1=2 UNION SELECT " ' AND 1=2 UNION SELECT @@version -- -" -- - |

|  |
| --- |
| <http://localhost:8080/NenthomWeb/servlets/ProductDetail_Servlet?productID=1%20UNION%20ALL%20SELECT%201,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20> |

|  |
| --- |
| <http://localhost:8080/NenthomWeb/servlets/SearchProduct_Servlet?search=%27+OR+%271%27+%3D+%271> |

Ta thấy rằng, sau khi thực hiện thủ công tiêm SQLi vào URL trên và tấn công thì hệ thống đều trả về thông báo lỗi bảo mật, do đó ta tiến hành tấn công vào các trang nghi ngờ khác.

##### 6.3.2.2.5. Nguyên nhân

Qua quá trình kiểm thử lỗ hổng SQL Injection bằng nhiều công cụ khác nhau như OWASP ZAP, SQLMap, jSQL Injection và phương pháp kiểm thử thủ công, có thể đưa ra nhận định rằng hệ thống hiện tại không tồn tại lỗ hổng SQL Injection tại các điểm được kiểm tra. Mặc dù công cụ ZAP phát hiện một số dấu hiệu nghi ngờ (possible SQL injection), tuy nhiên các công cụ kiểm thử chuyên sâu hơn như SQLMap và jSQL không thể khai thác thành công, đồng thời kiểm thử thủ công cũng không đem lại kết quả khai thác rõ ràng.

Nguyên nhân chính được xác định như sau:

Hệ thống đã sử dụng các cơ chế truy vấn an toàn, điển hình là việc áp dụng PreparedStatement (câu lệnh SQL có tham số hóa) trong các lớp xử lý dữ liệu (DAO). Kỹ thuật này giúp ngăn chặn việc chèn mã SQL vào trong truy vấn do dữ liệu đầu vào được biên dịch riêng biệt với cấu trúc câu lệnh SQL.

|  |
| --- |
| // Product DAO  // Lấy thông tin sản phẩm theo ID  **public** Product getProductById(**int** productId) {  Product product = **null**;  String sql = "SELECT ProductID, Name, Description, Price, Stock, ImageBase64 FROM Products WHERE ProductID = ?";  **try** (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {  preparedStatement.setInt(1, productId);  ResultSet resultSet = preparedStatement.executeQuery();  **if** (resultSet.next()) {  **int** id = resultSet.getInt("ProductID");  String name = resultSet.getString("Name");  String description = resultSet.getString("Description");  **double** price = resultSet.getDouble("Price");  **int** stock = resultSet.getInt("Stock");  String imageBase64 = resultSet.getString("ImageBase64");  product = **new** Product(id, name, description, price, stock, imageBase64);  }  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** product;  } |

Dữ liệu đầu vào từ người dùng được kiểm tra và xử lý phù hợp, có thể bao gồm việc kiểm tra kiểu dữ liệu, độ dài, hoặc lọc bỏ các ký tự đặc biệt, từ đó hạn chế khả năng chèn mã độc hại.

Cơ chế xử lý lỗi của hệ thống đã được cấu hình tốt, không tiết lộ thông tin nhạy cảm như câu lệnh SQL, cấu trúc cơ sở dữ liệu hay thông báo lỗi chi tiết, từ đó gây khó khăn cho việc khai thác dựa trên phản hồi (response-based injection).

Một số phản hồi bất thường từ phía máy chủ như mã lỗi HTTP 500 khi thử nghiệm các payload nguy hiểm cho thấy hệ thống đang áp dụng các biện pháp bảo vệ bổ sung như kiểm tra mẫu đầu vào (input validation), xử lý ngoại lệ nội bộ để giảm thiểu rủi ro bảo mật.

|  |
| --- |
| protected void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  String productIDStr = request.getParameter("productID");  int productID = 0;  try {  if (productIDStr == null || productIDStr.trim().isEmpty()) {  throw new IllegalArgumentException("Thiếu mã sản phẩm (productID).");  }  productID = Integer.*parseInt*(productIDStr);  if (productID <= 0) {  throw new IllegalArgumentException("Mã sản phẩm phải là số nguyên dương hợp lệ.");  }  } catch (NumberFormatException e) {  request.setAttribute("errorMessage", "Mã sản phẩm không hợp lệ, phải là số nguyên.");  request.getRequestDispatcher("/views/xss\_error.jsp").forward(request, response);  return;  } catch (IllegalArgumentException e) {  request.setAttribute("errorMessage", e.getMessage());  request.getRequestDispatcher("/views/xss\_error.jsp").forward(request, response);  return;  } |

Từ các phân tích trên, có thể kết luận rằng hệ thống hiện tại đã được thiết kế với các biện pháp phòng ngừa SQL Injection cơ bản và hiệu quả.

## 6.4. Nâng cấp lên Https

### 6.4.1. Giao thức HTTP là gì?

HTTP (Hypertext Transfer Protocol) là một giao thức mã nguồn mở được sử dụng để truyền tải siêu văn bản. Đây là giao thức chính được sử dụng trên mạng World Wide Web (WWW) nhằm chuyển các loại dữ liệu như văn bản, hình ảnh, âm thanh, và video giữa máy chủ web và trình duyệt của người dùng.

HTTP hoạt động dựa trên nền tảng của bộ giao thức TCP/IP (Transmission Control Protocol/Internet Protocol), là tập hợp các giao thức truyền thông tiêu chuẩn trên Internet. Trong đó, TCP đảm nhận vai trò kiểm soát việc truyền dữ liệu, còn IP chịu trách nhiệm định tuyến và định danh địa chỉ trên mạng.

Quá trình trao đổi dữ liệu qua HTTP được thực hiện theo mô hình Client-Server, nghĩa là máy khách (thường là trình duyệt) gửi yêu cầu đến máy chủ và nhận lại phản hồi. Khi người dùng nhập địa chỉ trang web, trình duyệt sẽ dùng hệ thống phân giải tên miền (DNS) để tìm địa chỉ IP của máy chủ, sau đó thiết lập kết nối và gửi yêu cầu truy cập. Máy chủ sẽ phản hồi bằng các tài nguyên tương ứng như nội dung trang HTML, hình ảnh, âm thanh hoặc video.

Tuy nhiên, một hạn chế lớn của HTTP là dữ liệu trao đổi giữa máy khách và máy chủ không được mã hóa. Điều này làm tăng nguy cơ rò rỉ thông tin quan trọng như mật khẩu hoặc dữ liệu cá nhân nếu bị kẻ xấu nghe lén (sniffing) trong quá trình truyền tải. Đây chính là lỗ hổng bảo mật cần lưu ý khi sử dụng HTTP.

### 6.4.2. Giao thức HTTPS là gì?

HTTPS (Hypertext Transfer Protocol Secure) là một phiên bản nâng cao về mặt bảo mật của giao thức HTTP, được thiết kế nhằm tăng cường an toàn cho việc truyền tải dữ liệu trên mạng. Mặc dù cách thức hoạt động cơ bản giống HTTP, HTTPS được tích hợp thêm các lớp mã hóa thông qua chứng chỉ bảo mật SSL (Secure Sockets Layer) hoặc TLS (Transport Layer Security).

Các chứng chỉ này cho phép mã hóa toàn bộ dữ liệu được gửi và nhận giữa trình duyệt của người dùng và máy chủ web, giúp bảo vệ thông tin nhạy cảm khỏi các mối đe dọa trên Internet. Quá trình mã hóa và giải mã được thực hiện thông qua cặp khóa công khai và khóa riêng, đảm bảo rằng chỉ bên nhận mới có thể giải mã thông tin được gửi.

HTTPS sử dụng kiến trúc bảo mật phi đối xứng gọi là PKI (Public Key Infrastructure), trong đó bao gồm một khóa công khai dùng để mã hóa dữ liệu và một khóa riêng để giải mã. Sự kết hợp này đảm bảo rằng dữ liệu không thể bị đọc trộm hoặc thay đổi trong quá trình truyền đi.

Thông qua cơ chế mã hóa này, HTTPS giúp giữ cho dữ liệu an toàn trong suốt quá trình truyền tải, giảm thiểu nguy cơ bị tin tặc tấn công, từ đó bảo vệ quyền riêng tư và thông tin cá nhân của người dùng khi truy cập các trang web.

### 6.4.3. Sự khác nhau giữa HTTP và HTTPS

#### 6.4.3.1. Mục đích sử dụng

HTTP thường được áp dụng cho các website không yêu cầu bảo mật cao, chẳng hạn như blog cá nhân, trang tin tức, hoặc các website cung cấp thông tin cơ bản.

Ngược lại, HTTPS lại phù hợp với những trang web cần đảm bảo tính riêng tư và an toàn dữ liệu, chẳng hạn như các nền tảng thương mại điện tử, mạng xã hội, hoặc các hệ thống tài chính và ngân hàng.

#### 6.4.3.2. Chứng chỉ bảo mật SSL

Một điểm khác biệt then chốt giữa HTTP và HTTPS nằm ở việc HTTPS sử dụng chứng chỉ bảo mật SSL hoặc TLS để mã hóa dữ liệu. Nhờ vào đó, HTTPS trở thành phiên bản an toàn hơn của HTTP. Trong bối cảnh Internet ngày càng phổ biến, nhu cầu bảo vệ thông tin cá nhân khi truy cập và trao đổi dữ liệu trên web là rất cần thiết.

Việc triển khai HTTPS không chỉ giúp người dùng cá nhân bảo mật thông tin, mà còn là tiêu chuẩn bắt buộc đối với các tổ chức, doanh nghiệp để xây dựng lòng tin với khách hàng. Các chứng chỉ SSL được quốc tế công nhận giúp mã hóa dữ liệu giữa trình duyệt và máy chủ, ngăn chặn việc bị đánh cắp bởi các đối tượng độc hại.

#### 6.4.3.3. Port trên HTTP và HTPPS

HTTP mặc định sử dụng cổng (port) 80 để truyền dữ liệu.

Trong khi đó, HTTPS sử dụng port 443 – đây là cổng được thiết kế để thiết lập kết nối mã hóa giữa client và server, giúp đảm bảo rằng dữ liệu không bị rò rỉ hay bị chỉnh sửa trong quá trình truyền.

#### 6.4.3.4. Mức độ bảo mật

HTTP không có cơ chế mã hóa hay xác thực, do đó dữ liệu có thể dễ dàng bị truy cập trái phép nếu bị nghe lén. Người dùng gần như không có cách nào để nhận biết hay phòng tránh việc thông tin bị đánh cắp qua giao thức này.

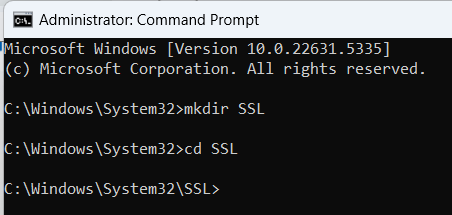
Trong khi đó, HTTPS cung cấp một mức độ bảo mật cao hơn nhiều. Khi truy cập một website sử dụng HTTPS, trình duyệt sẽ tiến hành xác minh danh tính của máy chủ thông qua chứng chỉ số (digital certificate) được cấp bởi các tổ chức chứng thực (CA – Certificate Authority). Những tổ chức này đóng vai trò như bên thứ ba đáng tin cậy, giúp xác thực thông tin và tăng cường an toàn cho người dùng khi truyền tải dữ liệu trên mạng.

### 6.4.4. Thiết lập HTTPS

Trước tiên, mở Command Prompt với quyền quản trị viên (Administrator)

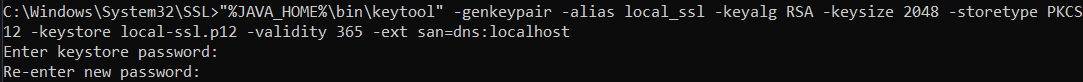
|  |
| --- |
|  |

Bước kế tiếp là tạo thư mục mang tên SSL bằng lệnh mkdir, và sau đó điều hướng vào thư mục vừa tạo bằng cách dùng lệnh cd SSL.



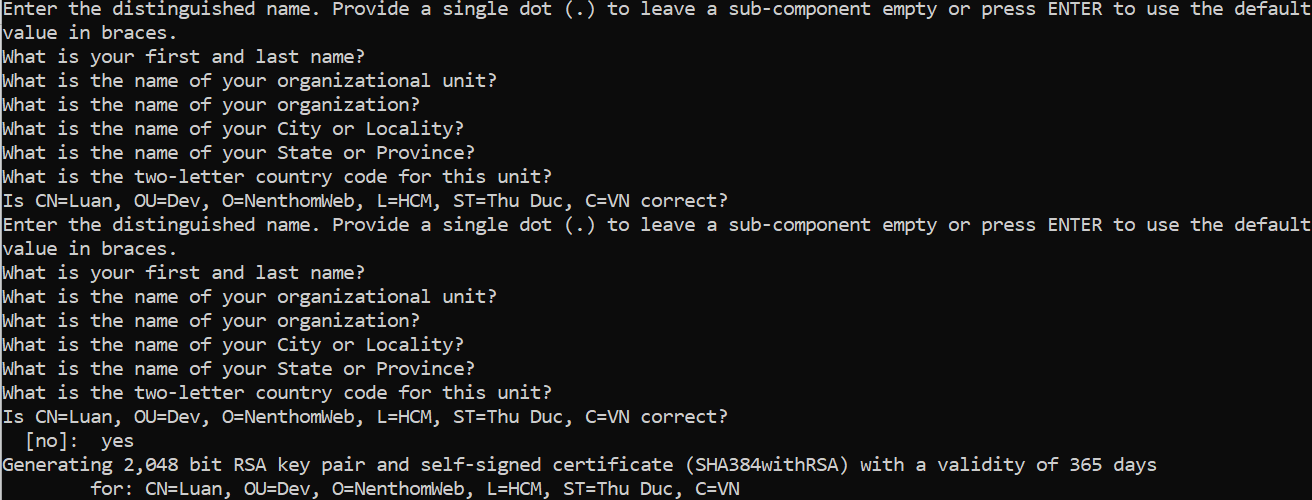
Sau bước đó, thực hiện nhập lệnh sau để khởi tạo một chứng chỉ SSL đáng tin cậy. Đây là một lệnh thuộc công cụ Java Keytool – một tiện ích dòng lệnh được dùng để quản lý kho khóa và chứng chỉ trong môi trường Java. Lệnh này sẽ tạo ra một cặp khóa công khai – bí mật (public/private key pair) cùng với chứng chỉ tự ký (self-signed certificate), rồi lưu toàn bộ vào một keystore theo chuẩn định dạng PKCS12.

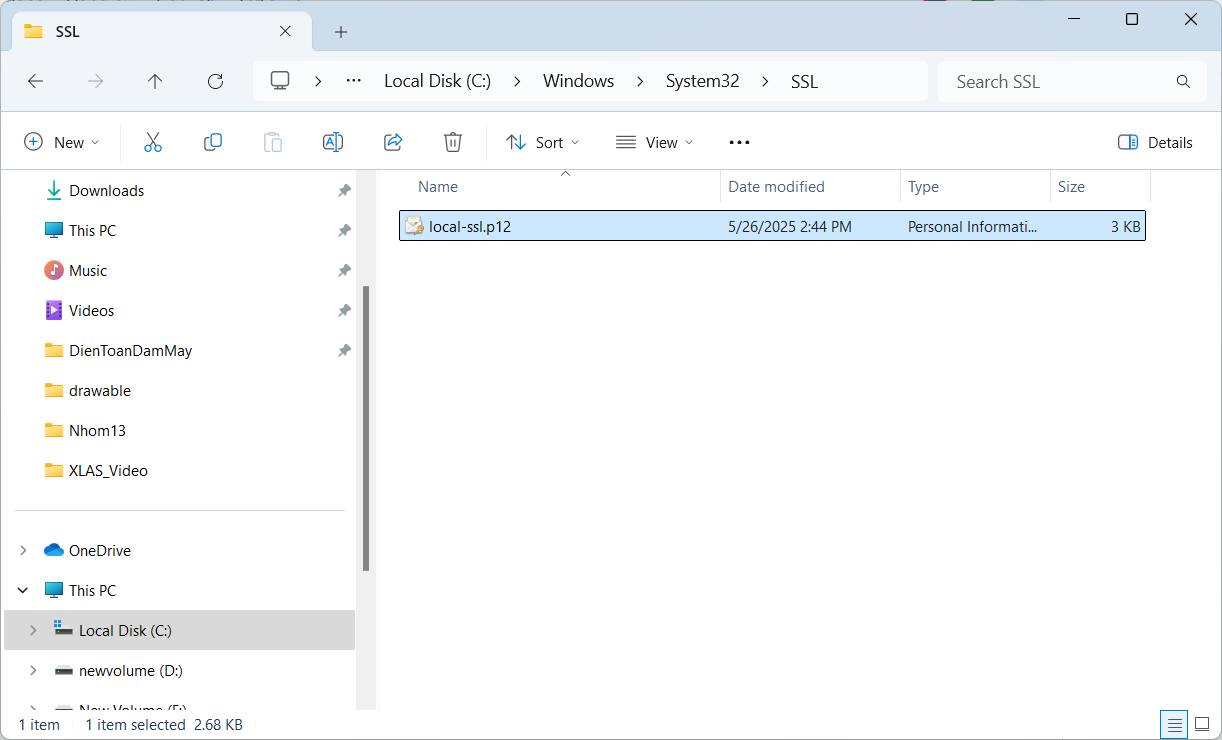
**"%JAVA\_HOME%\bin\keytool" -genkeypair -alias local\_ssl -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore local-ssl.p12 -validity 365 -ext san=dns:localhost**



Tiếp theo tiến hành nhập keystore password, và các thông số mà hệ thống yêu cầu như first name, last name, organization unit, organization, city, state, country code. Sau đó ấn yes để đồng ý với các input của chính bản thân mình.

Sau khi xác nhận, hệ thống sẽ tạo thành công một cặp khóa và chứng chỉ tự ký với thời hạn hiệu lực là 365 ngày.

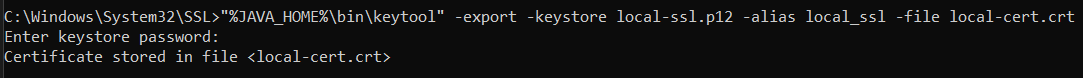


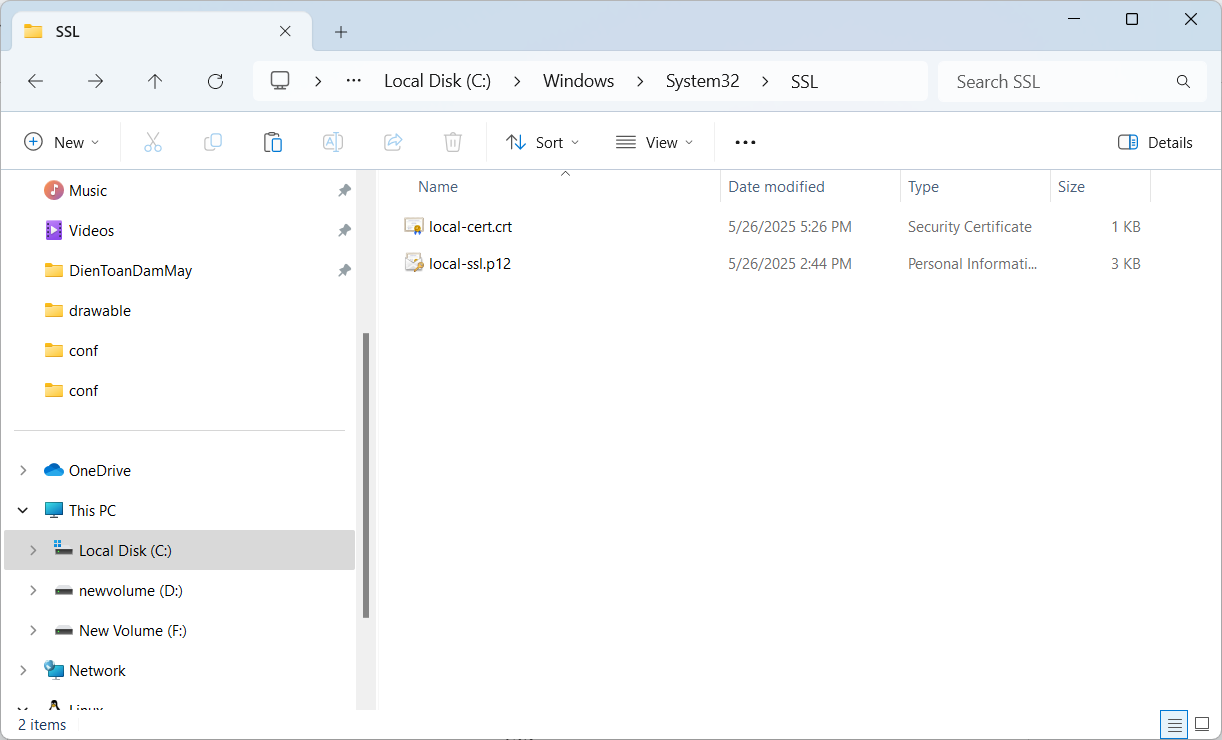


Tiếp tục mở lại Command Prompt và sử dụng lệnh sau để trích xuất chứng chỉ từ file keystore đã tạo trước đó.

**"%JAVA\_HOME%\bin\keytool" -export -keystore local-ssl.p12 -alias local\_ssl -file local-cert.crt**

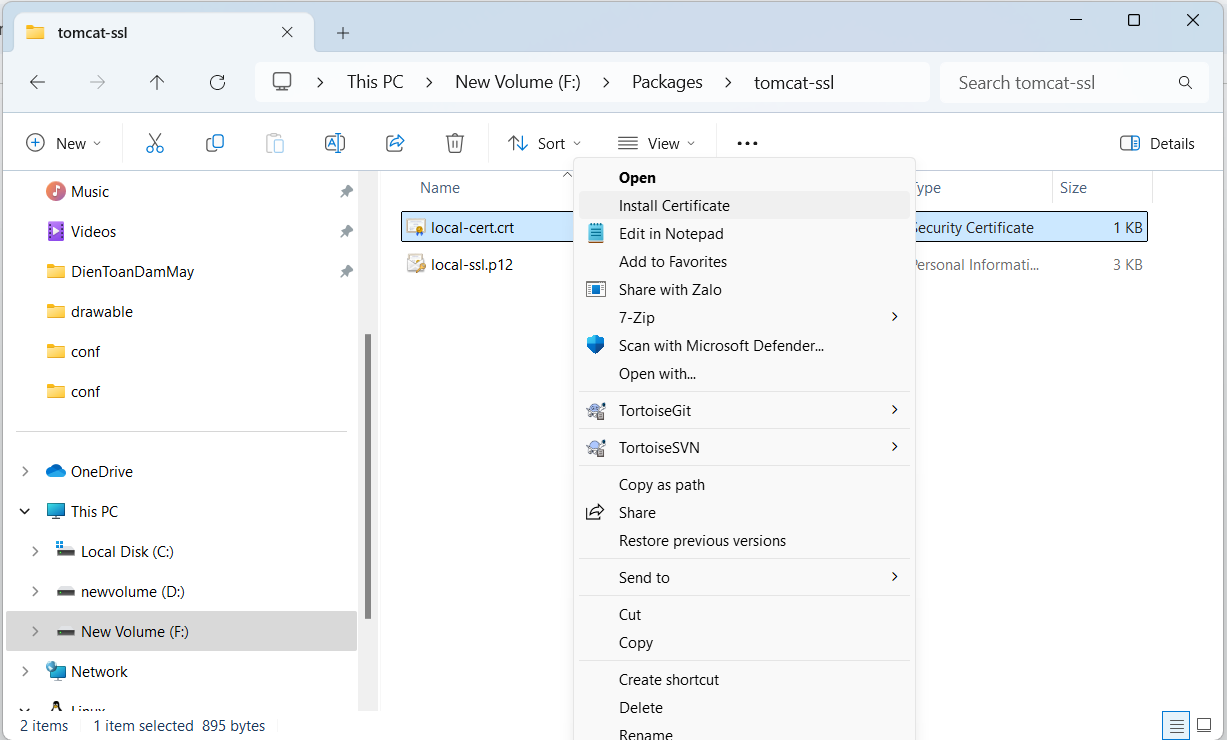
Sau khi thực thi lệnh, người dùng cần nhập lại mật khẩu keystore đã thiết lập trước đó và nhấn Enter. Nếu mật khẩu chính xác, hệ thống sẽ tiến hành tạo một chứng chỉ có tên là local-cert.



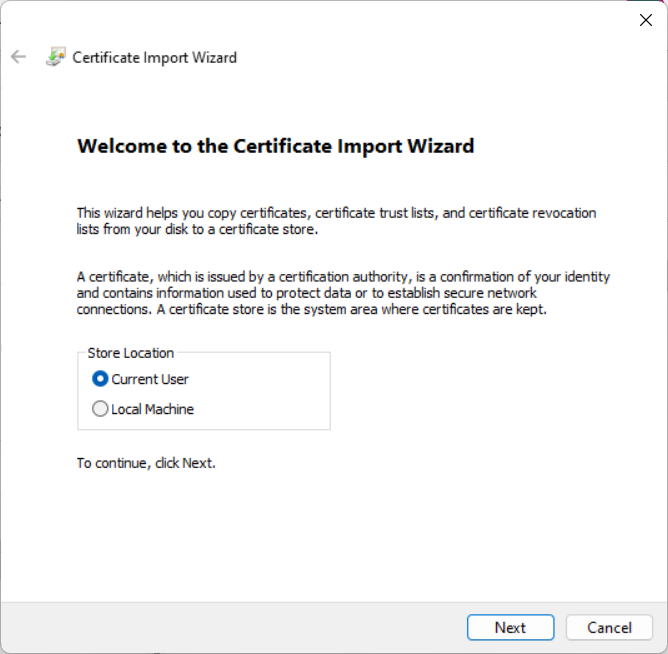


Chuyển 2 file sang ổ đĩa khác, vì ổ đĩa V thường yêu cầu quyền admin dẫn đến có thể bì từ chối truy cập

Chuột phải vào file local-cert.crt chọn Install Certificate

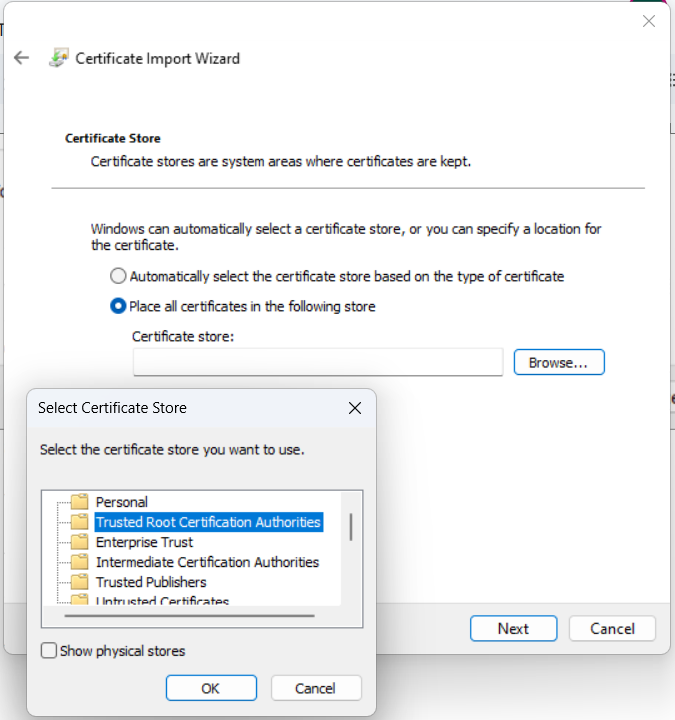


Vào lại thư mục SSL đó, chuột phải chọn Install Certificate

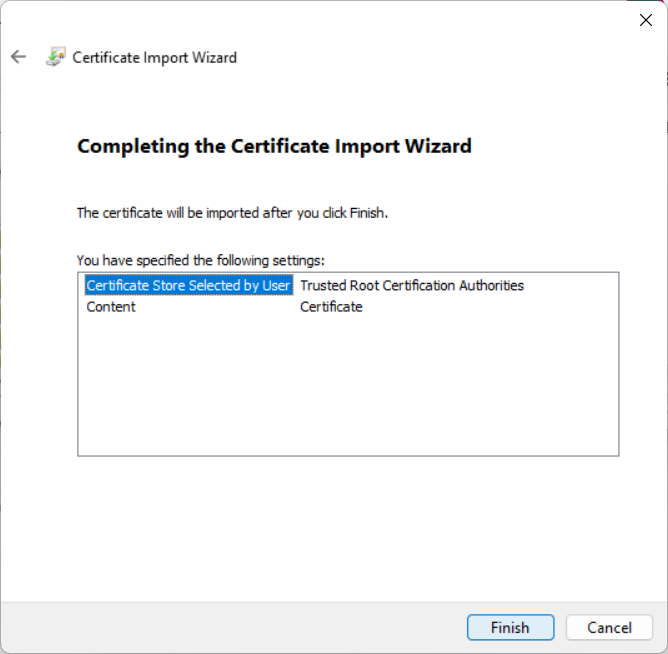


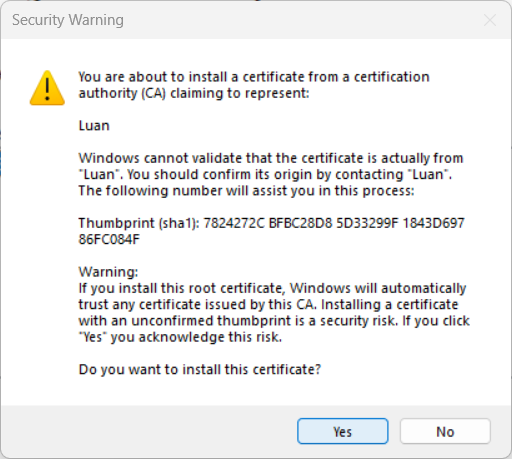
Vào lại thư mục SSL đó, chuột phải chọn Install Certificate

Chọn Trusted Root Certification Authorities và nhấn OK. Sau đó ta nhấn Next.

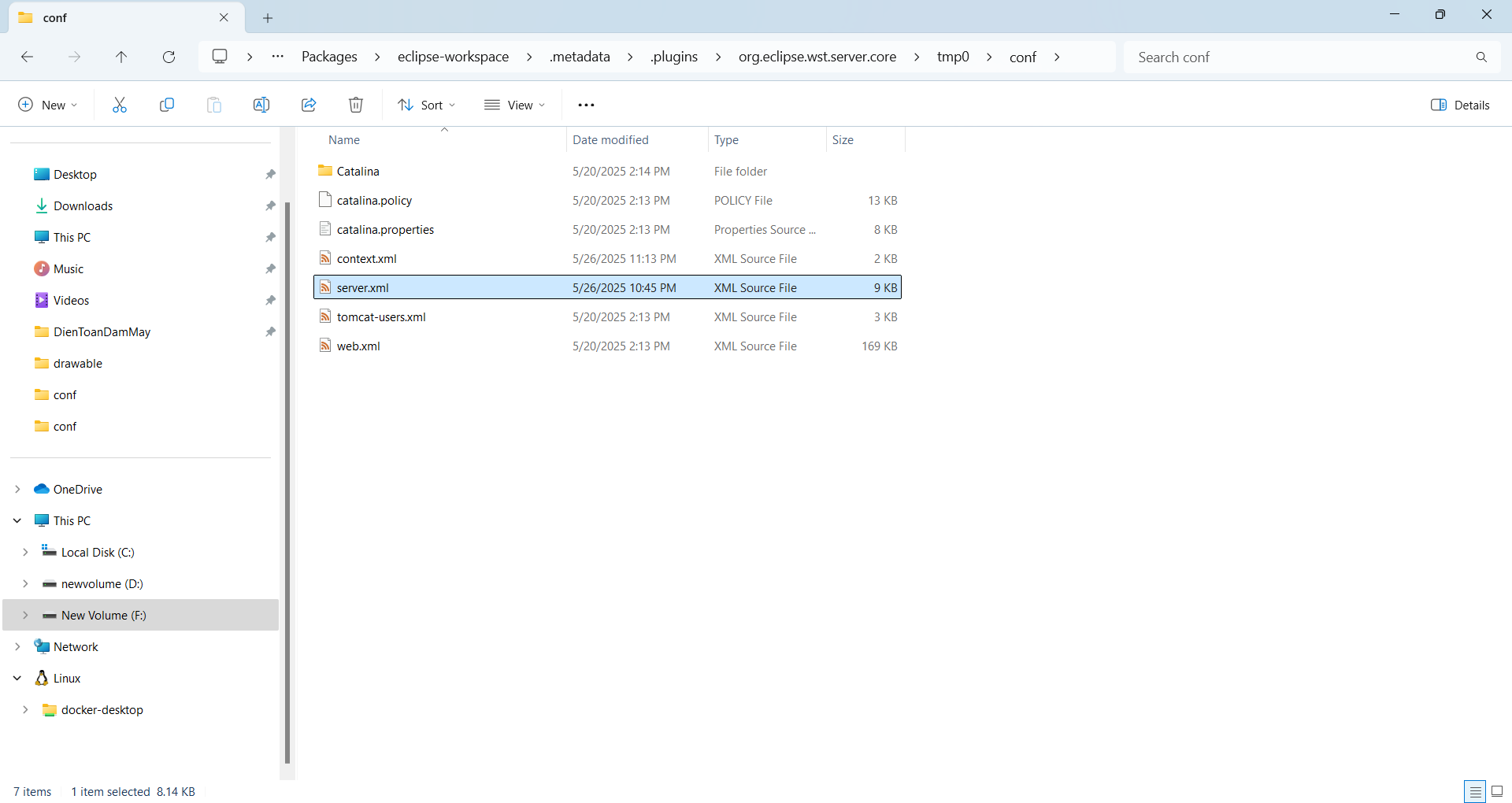


Cuối cùng ta ấn Finish và yes để kết thúc việc Install Certificates

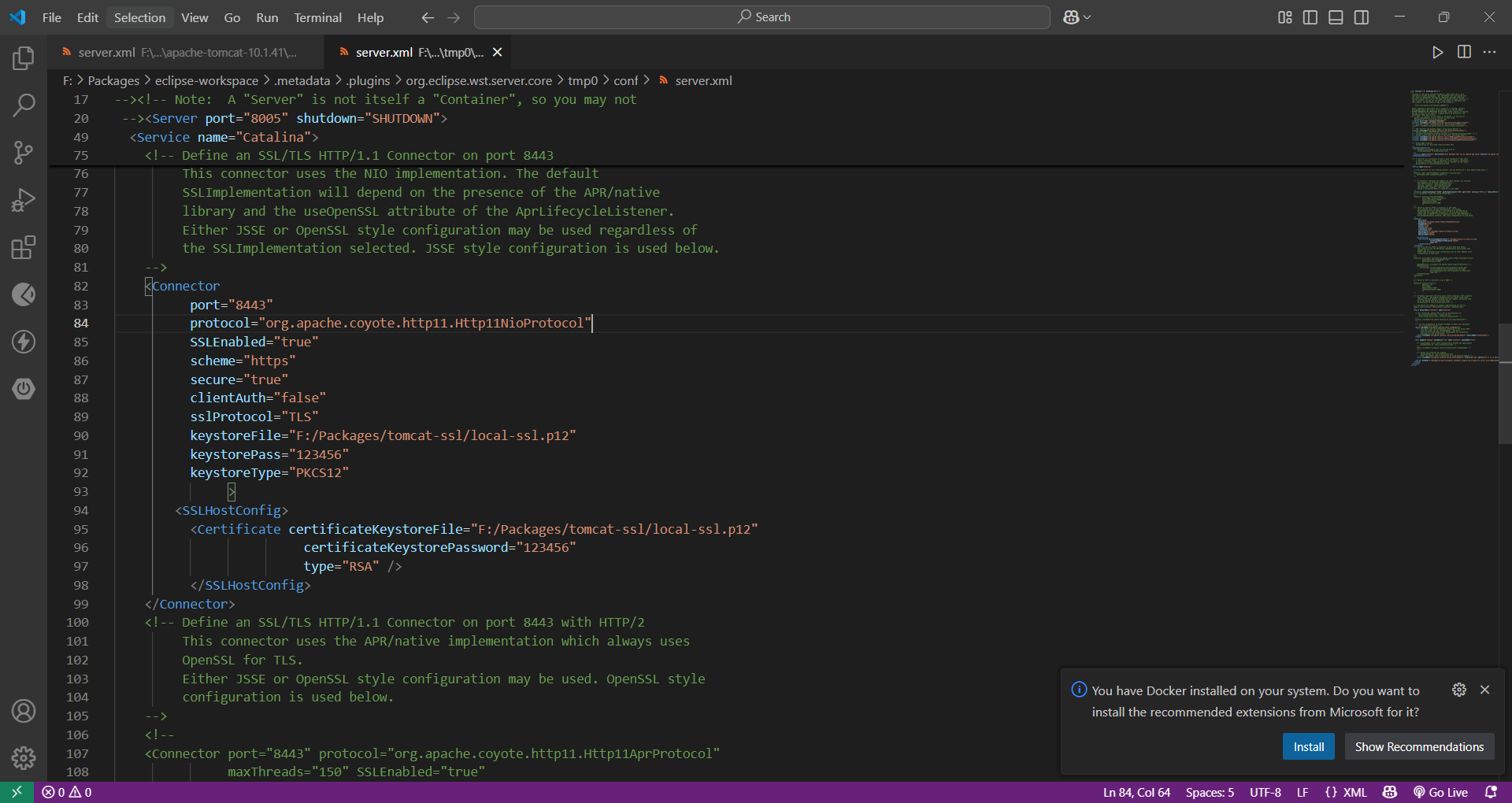




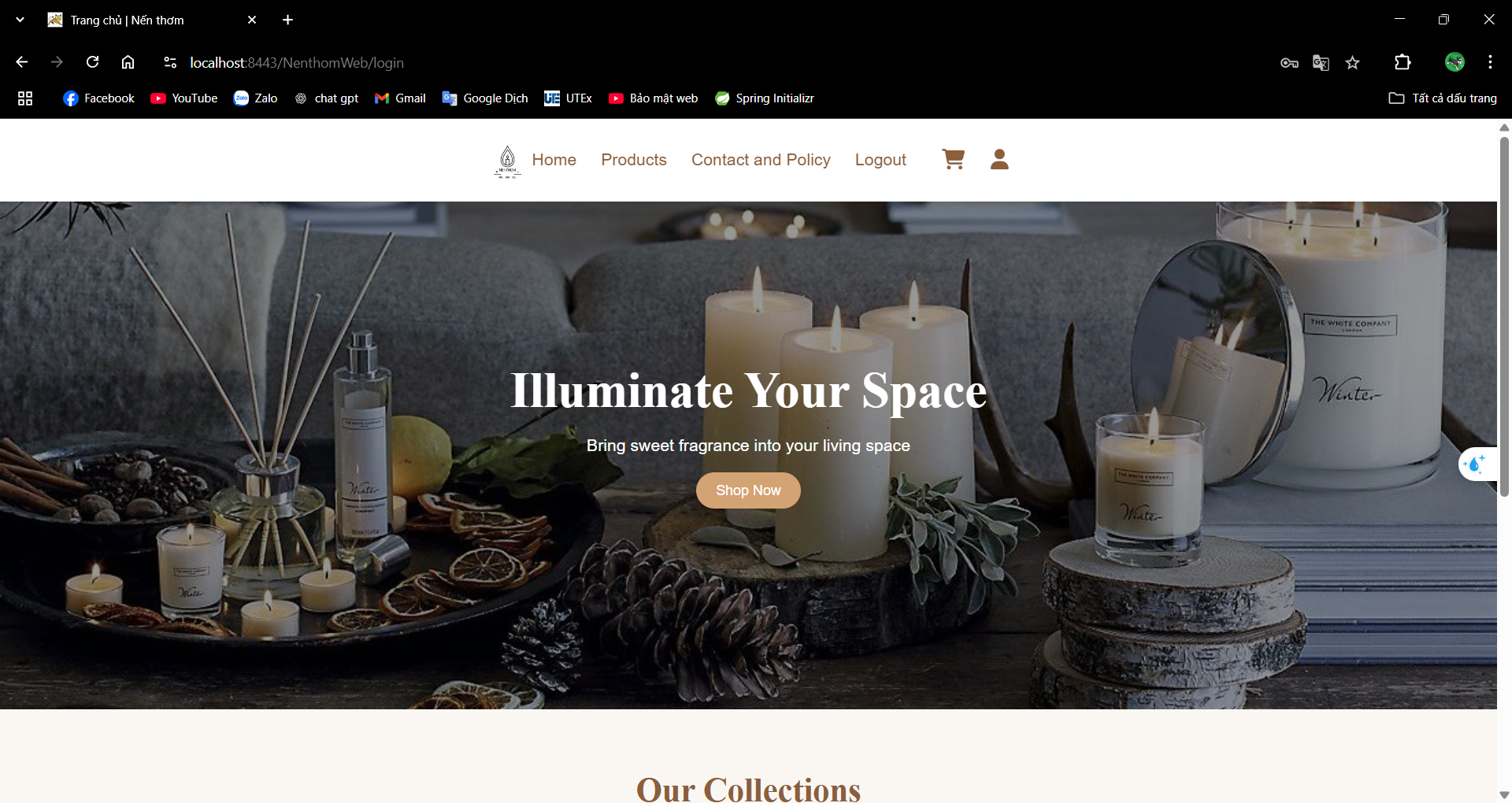
Chỉnh file server.xml trong thư mục .metadata của workspace Eclipse



Thêm đoạn sau vào trong thẻ <Service name="Catalina">:



Lên trình duyệt để xem có chạy được hay không



=> Vậy là ta đã nâng cấp trang web project từ HTTP lên HTTPS thành công

### 

# CHƯƠNG 7: BẢNG PHÂN CÔNG VÀ TỔNG HỢP LỖI

Bảng 7.1. Bảng phân công

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Huỳnh Minh Mẫn** | **Đỗ Phú Luân** | **Đăng Minh Nhật** | **Vi Quốc Thuận** | **Nguyễn Phúc Bảo** |
| Đặc tả phần mềm | Thiết kế cơ sở dữ liệu | Phần mở đầu  Thiết kế cơ sở dữ liệu  Cài đặt môi trường | Thiết kế giao diện | Cài đặt môi trường  Chạy chương trình |

Bảng 7.2. Bảng tổng hợp lỗi

|  |  |  |  |
| --- | --- | --- | --- |
| **Sinh viên thực hiện** | **Thủ công** | **Quét bằng ZAP** | **Lỗi hổng không tìm thấy** |
| Nguyễn Phúc Bảo | Buffer Overflow  Vulnerable and Outdated Components  Parameter Tampering | Content Security Policy (CSP) Header Not Set Cookie without SameSite Attribute |  |
| Đỗ Phú Luân | HTTPS | Absence of Anti-CSRF Tokens  X-Content-Type-Options Header Missing |  |
| Huỳnh Minh Mẫn | Broken Access Control | Cross-Domain Misconfiguration  Missing Anti-clickjacking Header |  |
| Đặng Minh Nhật | Brute Force Attack  Improper Input Validation |  | Insecure Direct Object Reference (IDOR) |
| Vi Quốc Thuận |  | Session ID in URL Rewrite  Cross Site Script (XSS) | SQL Injection |

# PHẦN KẾT LUẬN

Trong quá trình thực hiện đồ án xây dựng website bán nến thơm, nhóm đã không chỉ tập trung phát triển các tính năng phục vụ nhu cầu mua sắm trực tuyến mà còn đặc biệt chú trọng đến yếu tố bảo mật ứng dụng web – một yêu cầu thiết yếu trong môi trường số hiện nay.

Thông qua việc kết hợp giữa phương pháp kiểm thử thủ công và công cụ OWASP ZAP, nhóm đã tiến hành kiểm tra hệ thống một cách toàn diện, từ đó phát hiện ra nhiều lỗ hổng bảo mật tiềm ẩn. Các lỗi điển hình bao gồm: Brute Force Attack, Improper Input Validation, Insecure Direct Object Reference (IDOR), Buffer Overflow, Cross Site Scripting (XSS), Session ID in URL Rewrite, Absence of Anti-CSRF Tokens, Cookie without SameSite Attribute, Missing Anti-clickjacking Header, và các lỗi cấu hình như Content Security Policy Header Not Set, X-Content-Type-Options Header Missing, Cross-Domain Misconfiguration.

Tuy nhiên, một số lỗ hổng phổ biến như SQL Injection và IDOR không được phát hiện. Nguyên nhân có thể đến từ việc framework sử dụng đã tích hợp sẵn các cơ chế phòng chống, góp phần giảm thiểu nguy cơ khai thác.

Từ kết quả kiểm thử, nhóm nhận thức rõ rằng việc xây dựng một website không chỉ dừng lại ở mặt chức năng, giao diện người dùng mà còn phải đảm bảo an toàn thông tin và bảo mật dữ liệu cho người dùng. Việc phát hiện và hiểu rõ các lỗ hổng giúp nhóm có cái nhìn thực tế hơn về quy trình phát triển ứng dụng an toàn.

Kết luận, quá trình thực hiện đồ án không chỉ giúp nhóm củng cố kiến thức lập trình và thiết kế hệ thống, mà còn trang bị thêm kỹ năng kiểm thử và phát hiện lỗi bảo mật – một yếu tố quan trọng để phát triển các ứng dụng web tin cậy trong tương lai. Nhóm sẽ tiếp tục nghiên cứu sâu hơn về bảo mật web và áp dụng vào các dự án kế tiếp nhằm nâng cao chất lượng và độ an toàn cho người dùng.

# TÀI LIỆU THAM KHẢO

[1] *Testing for Insecure Direct Object References (IDOR)*. Truy cập từ <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/04-Testing_for_Insecure_Direct_Object_References>

[2] *Blocking Brute Force Attacks*. Truy cập từ <https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks>

[3] *Cross Site Scripting (XSS) Prevention Cheat Sheet*. Truy cập từ <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html>

[4] *Testing for SQL Injection (SQLi)*. Truy cập từ <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection>