**API**

API (Application Programming Interface), hay Giao diện Lập trình Ứng dụng, là cầu nối cho phép các ứng dụng, hệ thống hoặc dịch vụ khác nhau giao tiếp và trao đổi dữ liệu với nhau một cách hiệu quả. API hoạt động như một trung gian, cung cấp các điểm cuối (endpoints) để gửi yêu cầu và nhận phản hồi, giúp đơn giản hóa việc tích hợp và mở rộng chức năng của phần mềm.

**MỐI ĐE DỌA ĐỐI VỚI API**

Xác thực yếu

Tấn công tiêm mã (Injection Attacks)

Tấn công từ chối dịch vụ (Denial of Service - DoS)

**PHƯƠNG PHÁP AN TOÀN CHO API**

**1.JWT**

Khi đăng nhập thành công tạo ra 2 cookie là access\_token và refresh\_token.

TH1 : khi access\_token hết hạn mà refresh\_token còn hạn thì refresh\_token sẽ cấp 1 access\_token mới ...

**2.Rate Limiting**

1. Giới Thiệu

Cơ chế rate limiting (giới hạn tốc độ) là kỹ thuật bảo mật quan trọng để ngăn chặn tấn công brute force và bảo vệ tài nguyên server. Báo cáo này giải thích cơ chế rate limiting sử dụng database như đã triển khai trong hệ thống.

2. Cơ Chế Hoạt Động

2.1. Nguyên Lý Cơ Bản

Mục đích: Giới hạn số lượng yêu cầu từ một client trong khoảng thời gian xác định

Cách thức: Theo dõi và đếm các yêu cầu qua từng loại endpoint (login, API...)

2.2. Luồng Xử Lý

Nhận diện client: Sử dụng IP hoặc user ID làm định danh

Tạo khóa duy nhất: Kết hợp loại endpoint + định danh → hash SHA256

Kiểm tra trong database:

Nếu không có bản ghi hoặc đã hết hạn → tạo mới

Nếu còn hạn → tăng counter

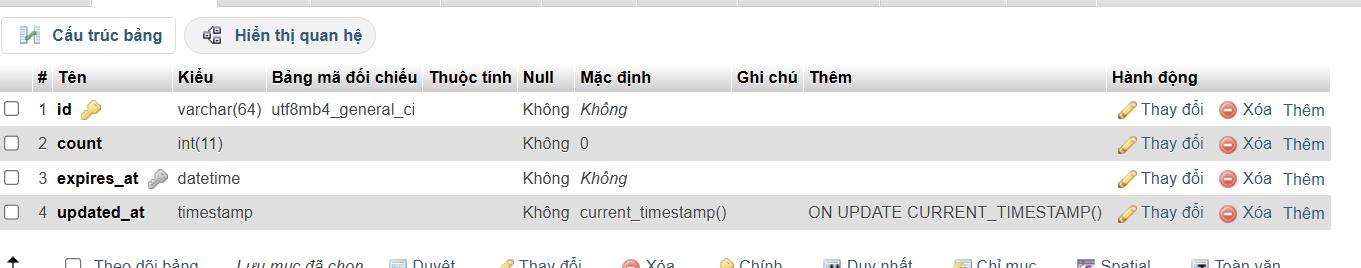
Quyết định:

Counter ≤ giới hạn → cho phép truy cập

Counter > giới hạn → trả về lỗi 429

3. Thiết Kế Database

3.1. Cấu Trúc Bảng rate\_limits



3.2. Tối Ưu Hiệu Năng

Indexing: Index cho id và expires\_at giúp truy vấn nhanh

Cleanup: Tự động xóa bản ghi hết hạn (chạy ngẫu nhiên 10% request)

4. Triển Khai Trong Ứng Dụng

4.1. Class DatabaseRateLimiter

4.2. Tích Hợp Vào API Login

5. Ưu Điểm Của Giải Pháp

5.1. Tính Bảo Mật

Hash key: Bảo vệ thông tin client

Phân biệt endpoint: Áp dụng giới hạn khác nhau cho từng loại API

IP-based: Ngăn chặn tấn công từ cùng IP

5.2. Tính Linh Hoạt

Dễ dàng điều chỉnh: Thay đổi limit/window trong config

Mở rộng: Thêm loại rate limit mới dễ dàng

Tương thích: Hoạt động với mọi hệ thống sử dụng PDO

5.3. Hiệu Suất

Cleanup thông minh: Chỉ chạy 10% request để giảm tải DB

Transaction: Sử dụng cơ chế atomic của database

1. **CSRF Token**

### ****Bảo vệ API khỏi tấn công CSRF bằng CSRF Token****

Để ngăn chặn tấn công Cross-Site Request Forgery (CSRF), hệ thống triển khai cơ chế **CSRF Token** như một lớp bảo mật bổ sung khi sử dụng **cookie để lưu Access Token**. Tấn công CSRF xảy ra khi kẻ tấn công lợi dụng trình duyệt đã đăng nhập của người dùng để gửi các yêu cầu độc hại đến server mà người dùng không hay biết.

Trong hệ thống, CSRF Token được tạo tại thời điểm người dùng đăng nhập thành công. Token này được lưu trữ đồng thời ở **session phía server** và **cookie phía client**. Tuy nhiên, để đảm bảo an toàn, frontend (JavaScript) sẽ chủ động **đọc token từ cookie và gửi lại token này trong mỗi request bảo mật thông qua một custom header** (ví dụ: X-CSRF-Token).

Tại phía server, mỗi request được kiểm tra bằng cách so sánh giá trị CSRF Token được gửi qua header với token đã lưu trong session. Nếu hai giá trị này không khớp, server sẽ từ chối xử lý request.

Việc yêu cầu token phải được gửi qua **hai kênh độc lập** (cookie và header) giúp chống lại các request giả mạo từ các trang web bên ngoài. Hacker có thể khiến trình duyệt tự gửi cookie, nhưng **không thể chèn token hợp lệ vào header**, vì JavaScript trên trang web bên ngoài không có quyền truy cập cookie của domain khác. Nhờ đó, CSRF Token giúp đảm bảo rằng request chỉ đến từ **giao diện người dùng hợp pháp của hệ thống**.

· **Sử dụng HTTPS**:

* ·

Đảm bảo rằng tất cả các yêu cầu đến API đều được gửi qua HTTPS thay vì HTTP. Điều này giúp bảo vệ dữ liệu truyền tải khỏi việc bị nghe lén (Man-in-the-Middle attack).

· **Cung cấp xác thực API mạnh mẽ**:

* ·

**JWT (JSON Web Tokens)**: Sử dụng JWT để xác thực người dùng khi truy cập API. Mỗi API request sẽ phải kèm theo một token hợp lệ, giúp bảo vệ API khỏi việc truy cập trái phép.

**OAuth2**: Sử dụng OAuth2 cho các API cần tích hợp với các dịch vụ khác (ví dụ, đăng nhập qua Google hoặc Facebook).

· **Cài đặt Rate Limiting**:

* ·

Giới hạn số lần mà một IP hoặc tài khoản có thể gọi API trong một khoảng thời gian. Điều này giúp ngăn ngừa các cuộc tấn công DoS/DDoS hoặc brute force.

· **Xác thực quyền truy cập (Authorization)**:

* ·

Kiểm tra quyền truy cập của người dùng trước khi cho phép họ thực hiện các hành động. Ví dụ: chỉ những người có quyền admin mới có thể xóa dữ liệu.

· **Kiểm tra và lọc đầu vào**:

* ·

**Validation**: Luôn kiểm tra và xác thực dữ liệu đầu vào để tránh SQL Injection, XSS, hoặc các tấn công khác.

**Sanitization**: Làm sạch dữ liệu đầu vào (ví dụ: loại bỏ các ký tự đặc biệt không cần thiết).

· **Sử dụng API Gateway**:

* ·

Đặt một API Gateway giữa ứng dụng của bạn và các API backend để quản lý tất cả các yêu cầu, xác thực, rate limiting và bảo vệ chống lại các cuộc tấn công.

· **Đảm bảo rằng API không tiết lộ thông tin nhạy cảm**:

* ·

Đảm bảo rằng các lỗi và thông báo API không tiết lộ chi tiết về cấu trúc hệ thống hoặc cơ sở dữ liệu của bạn. Các thông báo lỗi nên rõ ràng, nhưng không quá chi tiết.