|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  logo_128  ĐỒ ÁN  **THỰC TẬP TẠI SVMC**  **ĐỀ TÀI:**  **High availability API gateway as platform service**   |  |  | | --- | --- | | **Sinh viên thực hiện:** | Nguyễn Thị Hải - 187715051 | |  | Cao Văn Duy - 001098010785 | | **Mentor hướng dẫn:** | Anh Hà Văn Đạt |   Hà Nội, 6-2021 |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  logo_128  ĐỒ ÁN  **THỰC TẬP TẠI SVMC**  **ĐỀ TÀI:**  **High availability API gateway as platform service**   |  |  | | --- | --- | | **Sinh viên thực hiện:** | Nguyễn Thị Hải | |  | Cao Văn Duy | | **Mentor hướng dẫn:** | Anh Hà Văn Đạt | |  |  |   Hà Nội, 6-2021 |

# LỜI NÓI ĐẦU

Hiện nay, hệ thống IoT nói chung và kiến trúc Microservices đang được áp dụng rộng rãi trong các hệ thống lớn. Kiến trúc Microservice là một trong những xu hướng kiến trúc phần mềm được sử dụng nhiều nhất tại thời điểm này và nó đã thay đổi cách thức xây dựng hệ thống theo những công nghệ của thập kỷ trước. Bên cạnh đó, hệ thống đòi hỏi phải luôn trong tình trạng hoạt động và mang tính sẵn sàng cao. Do vậy, em đã chọn đề tài “**High availability API gateway as platform service**” cho đồ án thực tập của mình. Đồ án thực tập này của emđược nghiên cứu và triển khai tại bộ phận platform service, SVMC.

Trong quá trình thực hiện đề tài, em đã cố gắng để đạt được kết quả tốt nhất, tuy nhiên vẫn không thể tránh khỏi những thiếu sót và một số nhiệm vụ chưa hoàn thành. Vì vậy em rất mong nhận được các ý kiến đóng góp của các anh chị để đồ án của em thêm hoàn thiện và có thêm được những kinh nghiệm quý báu cho bản thân phục vụ cho quá trình học tập và làm việc trong tương lai.

Em xin gửi lời cảm ơn đến anh Hà Văn Đạt và các anh chị trong công ty SVMC đã giúp đỡ em hoàn thành đề tài này.

# MỤC LỤC

[DANH DÁCH CÁC TỪ VIẾT TẮT 1](#_Toc73916723)

[DANH MỤC HÌNH VẼ 2](#_Toc73916724)

[CHƯƠNG 1 MỞ ĐẦU 3](#_Toc73916725)

[1.1 Đặt vấn đề 3](#_Toc73916726)

[1.2 Ý tưởng đề tài 3](#_Toc73916727)

[1.3 Phạm vi nghiên cứu 3](#_Toc73916728)

[CHƯƠNG 2 API GATEWAY 5](#_Toc73916729)

[2.1 Kiến trúc Microservice 5](#_Toc73916730)

[2.2 API Gateaway là gì? 6](#_Toc73916731)

[2.3 Lợi ích của API Gateway 6](#_Toc73916732)

[2.4 Nhược điểm của API Gateway 7](#_Toc73916733)

[CHƯƠNG 3 TÍNH SẴN SÀNG CAO CỦA API GATEWAY 8](#_Toc73916734)

[3.1 Tính sẵn sàng cao (high avalability) 8](#_Toc73916735)

[3.2 Vai trò Load balancing trong hệ thống 10](#_Toc73916736)

[3.3 Một số thuật toán cân bằng tải (Load balancing) 10](#_Toc73916737)

[3.3.1 Round Robin 10](#_Toc73916738)

[3.3.2 Weighted Round Robin 12](#_Toc73916739)

[3.3.3 Least connected 12](#_Toc73916740)

[3.3.4 Session persistence – Phiên hoạt động 13](#_Toc73916741)

[3.3.5 Least Time 13](#_Toc73916742)

[CHƯƠNG 4 PHÁT TRIỂN PHẦN MỀM 14](#_Toc73916743)

[4.1 Giới thiệu và cài đặt JHipter 14](#_Toc73916744)

[4.2 Giới thiệu và cài đặt Nginx 14](#_Toc73916745)

[4.2.1 Giới thiệu Nginx 14](#_Toc73916746)

[4.2.2 Nguyên lý hoạt động Nginx 14](#_Toc73916747)

[4.2.3 Cài đặt Nginx 15](#_Toc73916748)

[4.3 Phát triển phần mềm trên Jhipster và nginx 16](#_Toc73916749)

[4.3.1 Mô hình hệ thống Microservices trên Jhipster 16](#_Toc73916750)

[4.3.2 Tạo Gateway trên Jhipster 16](#_Toc73916751)

[4.3.3 Tạo Microservice trêm JHipter 17](#_Toc73916752)

[4.3.4 Tạo entity trên JHipter 18](#_Toc73916753)

[4.3.5 Sử dụng NGINX làm load balancer 19](#_Toc73916754)

[CHƯƠNG 5 KẾT QUẢ VÀ KIỂM THỬ 21](#_Toc73916755)

[5.1 Kịch bản demo 21](#_Toc73916756)

[5.2 Kết quả đạt được 21](#_Toc73916757)

[CHƯƠNG 6 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 24](#_Toc73916758)

[DANH MỤC TÀI LIỆU THAM KHẢO 25](#_Toc73916759)

# DANH DÁCH CÁC TỪ VIẾT TẮT

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Thuật ngữ viết tắt** | **Thuật ngữ Tiếng anh** | **Thuật ngữ Tiếng việt** |
| 1 | HA | High Availability | Tính sẵn sàng (khả dụng) cao |
| 2 | API | Application Programming Interface | Giao diện lập trình ứng dụng |
| 3 | LB | Load Balancer | Cân bằng tải |
| 4 | BE | Back end |  |

# DANH MỤC HÌNH VẼ

[Hình 2‑1 Kiến trúc Microservices không sử dụng và có sử dụng API Gateway [1] 6](#_Toc73915974)

[Hình 3‑1 Sơ đồ hệ thống API gateway mang tính sẵn sàng cao [2] 9](#_Toc73915975)

[Hình 3‑2 Thuật toán Round Robin 11](#_Toc73915976)

[Hình 3‑3 Thuật toán Least connected 12](#_Toc73915977)

[Hình 4‑1 Nguyên lý hoạt động Nginx [4] 15](#_Toc73915978)

[Hình 4‑2 Mô hình hệ thống Microservices trên JHipster [3] 16](#_Toc73915979)

[Hình 4‑3 Tạo Gateway trên Jhipster 17](#_Toc73915980)

[Hình 4‑4 Tạo Microservice trên JHipster 18](#_Toc73915981)

[Hình 4‑5 Tạo entity Customer trên JHipster 19](#_Toc73915982)

[Hình 5‑1 Giao diện quản lý JHipster Registry 21](#_Toc73915983)

[Hình 5‑2 Load balancer điều hướng đến Gateway\_1 22](#_Toc73915984)

[Hình 5‑3 Load balancer điều hướng đến Gateway\_2 22](#_Toc73915985)

[Hình 5‑4 Giao diện quản lý entity 23](#_Toc73915986)

[Hình 5‑5 gọi BE API qua cổng Nginx 23](#_Toc73915987)

# MỞ ĐẦU

Chương này nêu ra các vấn đề mà đồ án cần giải quyết, trình bày ý tưởng thực hiện đề tài, giới hạn phạm vi nghiên cứu của đề tài.

## Đặt vấn đề

Xu hướng thương mại điện tử ngày càng phát triển mạnh mẽ trên toàn cầu. Các hệ thống ngày càng lớn và phức tạp hơn bao gồm: dữ liệu khách hàng, dữ liệu sản phẩm, phương tiện truyền thông… Người dùng có thể truy cập đến từ nhiều nguồn khác nhau như máy tính, điện thoại, website… Đồng nghĩa với số lượng request lớn đòi hỏi hệ thống phải đáp ứng nhanh chóng, chính xác và kịp thời. Vì vậy, các máy chủ phải luôn trong tình trạng “sẵn sàng cao”, sẵn sàng phục vụ và giảm thiểu khả năng gián đoạn của hệ thống. Trong đề tài này, em đã xây dựng và phát triển hệ thống API Gateway mang tính sẵn sàng cao làm dịch vụ nền tảng. Từ đó giúp tiếp cận các xu hướng công nghệ hiện nay.

## Ý tưởng đề tài

Một trong những lợi ích của việc thiết kế hệ thống theo kiến trúc microservies sao với kiến trúc nguyên khối (monolithic) là khả năng mở rộng, phát triển thêm các microservice riêng lẻ. Thời gian hoạt động liên tục (uptime) luôn bị giới hạn bởi lượng truy cập. Nếu một hệ thống sử dụng một cổng API Gateway làm điểm truy cập duy nhất cho client, điều gì sẽ xảy ra nếu điểm này có vấn đề? Cả một hệ thống sẽ bị “chết”. Để giải quyết vấn đề đó, API Gateway phải luôn trong chế độ “sẵn sàng cao” (high availability)

## Phạm vi nghiên cứu

Trong đề tài này, nhằm đảm bảo hệ thống Microservice không xảy ra một điểm lỗi nà, client không kết nối trực tiếp vào Gateway API mà được kết nối qua một server đóng vai trò load balancing. Điều này giúp loại bỏ bất kỳ thời gian ngừng hoạt động tiềm ẩn nào của hệ thống trong quá trình vận hành. Đóng góp về mặt kỹ thuật của đồ án này bao gồm:

* Xây dựng hệ thống theo kiến trúc Microservice mà trong đó API Gateway được đặt trong chế độ HA (High Availability)
* Đưa ra giải pháp load balancing cho hệ thống

# API GATEWAY

Ở chương này em sẽ giới thiệu cũng như đi sâu vào việc nghiên cứu lý thuyết vế đầu tiên của đề tài: các khái niêm tổng quan về API gateway cũng như lợi ích và những nhược điểm của hệ thống sử dụng API gateway

## Kiến trúc Microservice

Với kiến trúc Microservices mỗi dịch vụ sẽ được chia nhỏ thành nhiều thành phần khác nhau, mỗi thành phần sẽ hoạt động độc lập, được phát triển độc lập và chỉ xử lý các nghiệp vụ chức năng của nó. Mỗi thành phần cũng sẽ không lệ thuộc vào công nghệ phát triển với các thành phần khác.

Ưu điểm của kiến trúc Microservice:

* Kiến trúc phù hợp cho các dịch vụ lớn. Chia nhỏ các chức năng thành các thành phần, lập trình viên thuận lợi hơn trong việc phát triển, vận hành mỗi thành phần độc lập với nhau
* Thuận lợi trong việc phát triển lâu dài cho dịch vụ. Vì mỗi thành phần chỉ thực hiện đúng một chức năng, nên việc cải tiến sẽ dễ dàng thực hiện và không gây gián đoạn tới các thành phần khác. Mỗi thành phần sẽ giao tiếp với nhau qua giao diện riêng
* Việc phát triển mỗi thành phần của dịch vụ cũng được chia nhỏ, và đưa tới các team phát triển, khối lượng việc xử lý của mỗi team sẽ nhỏ và nhanh chóng hơn
* Mỗi thành phần sẽ không phụ thuộc vào công nghệ với các thành phần khác
* Với kiến trúc Monolithic khi một chức năng trong dịch vụ gặp lỗi, sẽ dẫn đến toàn bộ dịch vụ bị gián đoạn. Với kiến trúc Microservices, khi một thành phần gặp lỗi, các thành phần khác sẽ không bị gián đoạn

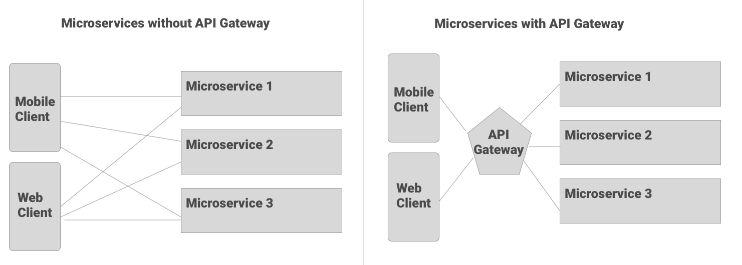
Bên cạnh các ưu điểm, kiến trúc Microservice cũng có một số nhược điểm như sau:

* Dịch vụ sẽ có nhiều thành phần phải quản lý
* Các yêu cầu về hạ tầng để đáp ứng cho các thành phần của dịch vụ cũng sẽ phức tạp hơn
* Dịch vụ sẽ được phát triển bởi nhiều công nghệ, ngôn ngữ lập trình
* Việc kiểm thử toàn bộ hệ thống sẽ phức tạp hơn

## API Gateaway là gì?

Trên lý thuyết, client có thể giao tiếp trực tiếp với services trong mô hình Microservices. Với những hê thống nhỏ có một hoặc hai API thì không gặp vấn đề gì lớn. Tuy nhiên, với các hệ thống lớn được xây dựng bởi nhiều API và nhiều client khác nhau thì gặp một số vấn đề sau:

* Client phải kết nối với nhiều endpoints
* Xác thực và phân quyền (authentication and authentication) cho từng endpoints
* Vấn đề security bị hạn chế: client biết được hệ thống có bao nhiêu service
* Tổng hợp dữ liệu như thế nào từ nhiều API khác nhau?
* Quản lý services như thế nào hoặc muốn chặn một địa chỉ IP nào đó thì phải chặn ở từng service



Hình 2‑1 Kiến trúc Microservices không sử dụng và có sử dụng API Gateway [1]

Các vấn đề trên được giải quyết bằng cách xây dựng một cổng trung gian gọi là API Gateway. API gateway là cổng kết nối API đứng giữa client và backend services. API Gateway được xem như Reverse Proxy forward từng request đến API tương ứng. Client chỉ cần kết nối đến API gateway mà không cần bên trong nó có bao nhiêu service.

## Lợi ích của API Gateway

* Che dấu backend service phía bên trong mà chỉ đưa ra API Gateway duy nhất

Clients sẽ tương tác với hệ thống của chúng ta thông qua api gateway chứ không gọi trực tiếp tới một services cụ thể, các endpoints của các services sẽ chỉ được gọi nội bộ, tức là gọi giữa các services với nhau hoặc được gọi từ API gateway, người dùng sẽ gọi các api này thông qua các public endpoints từ API Gateway. Chính vì vậy cho nên phía client không cần và cũng không thể biết được các services phía backend được phân chia như thế nào, việc refactor code frontend cũng dễ dàng hơn đối với lập trình viên.

* Phần code phía frontend sẽ gọn gàng hơn

Vì không phải tracking nhiều endpoints, tất cả chỉ việc gọi đến api gateway nên phần code frontend sẽ gọn gàng hơn so với việc phải tracking hàng tá endpoints trên từng services một, nhất là khi mà hệ thống ngày một phình to ra.

* Vấn đề authentication services

API gateway thường cung cấp nhiều cơ chế xác thực, chúng ta có thể sử dụng nó để xác thực người dùng luôn, giúp tiết kiệm thời gian và làm hệ thống chúng ta đơn giản hơn.

* Dễ dàng quản lý và giám sát API

Giả sử có một địa chỉ IP gửi request đến cùng một API nhiều lần, API Gateway có thể giúp ngăn chặn những địa chỉ IP đó.

* API Gateway hỗ trợ tổng hợp Data bằng cách gọi một lần thay vì gọi lần lượt

## Nhược điểm của API Gateway

* Tăng thời gian response

Vì phải đi qua server trung gian cho nên việc response sẽ bị trễ hơn so với việc gọi trực tiếp tới hệ thống.

* API Gateway trở thành nút thắt cổ chai (bottleneck)

Hệ thống có nhiều service nhưng có một service bị chết hoặc rơi vào exception làm chậm cả hệ thống. Vì vậy phải quản lý lỗi của từng microservice để đảm bảo một service chết thì cả hệ thống không được chết.

* Tốn thêm chi phí

Các yêu cầu về hạ tầng cũng như quản lý hệ thống api gateway để đáp ứng cho các thành phần của dịch vụ cũng sẽ phức tạp hơn

# TÍNH SẴN SÀNG CAO CỦA API GATEWAY

## Tính sẵn sàng cao (high avalability)

Là một thành phần quan trọng và là điểm vào DUY NHẤT trong kiến trúc microservice, bạn nên triển khai API Gateway ở chế độ khả dụng Cao (HA). Các phiên bản API Gateway thường được triển khai đằng sau bộ cân bằng tải tiêu chuẩn, định kỳ truy vấn trạng thái của API Gateway. Nếu một vấn đề xảy ra, bộ cân bằng tải chuyển hướng lưu lượng truy cập đến các phiên bản hot stand-by (chờ nóng).

Sự kiện/cảnh báo được định cấu hình để nhận thông báo trong trường hợp bất kỳ vấn đề nào xảy ra. Nếu một sự kiện hoặc cảnh báo được kích hoạt, vấn đề có thể được xác định bằng cách sử dụng API Gateway Analytics và API Gateway đang hoạt động sau đó có thể được sửa chữa.

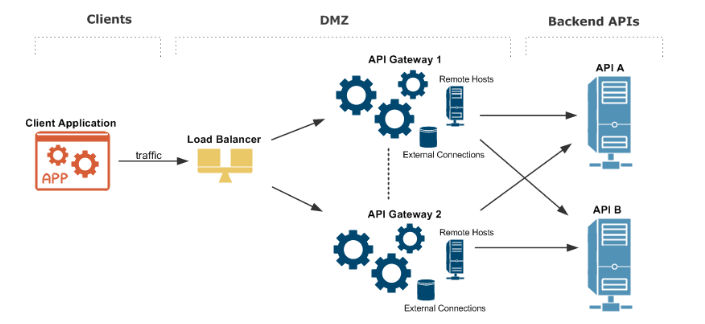
Bản chất các phiên bản API Gateway là không trạng thái. Không có dữ liệu phiên nào được tạo và do đó không cần sao chép trạng thái phiên trên các API Gateway. Tuy nhiên, API Gateway có thể duy trì dữ liệu được lưu trong bộ nhớ cache, dữ liệu này có thể được sao chép bằng cách sử dụng mối quan hệ ngang hàng, trên một cụm API Gateway.

Tính khả dụng cao có thể được duy trì bằng cách sử dụng một trong các tùy chọn sau: Active/Active, Active/Standby hoặc Active/Active. Chúng được mô tả như sau:

* Active/Standby: Hệ thống bị tắt
* Active/Passive: Hệ thống đang hoạt động nhưng không chứa trạng thái
* Active/Active: Hệ thống hoạt động hoàn toàn và ở trạng thái hệ thống hiện tại

Hướng dẫn về HA và chuyển đổi dự phòng

* Để đạt được tính khả dụng tối đa, API Gateway nên được sử dụng trong Active/Active cho mỗi API Gateway.
* Phân tích lưu lượng thích hợp, để giới hạn lưu lượng truy cập vào các dịch vụ phụ trợ, để bảo vệ khỏi tình trạng tràn ngập tin nhắn (message flooding). Điều này đặc biệt quan trọng với các hệ thống cũ gần đây đã được kích hoạt dịch vụ. Các hệ thống cũ có thể không được thiết kế cho các kiểu lưu lượng mà chúng đang phải chịu.
* Theo dõi cơ sở hạ tầng mạng một cách cẩn thận, để sớm xác định bất kỳ vấn đề nào. Bạn có thể thực hiện việc này bằng cách sử dụng API Gateway Manager và API Gateway Analytics. Các giao diện cũng được cung cấp cho các công cụ giám sát tiêu chuẩn, chẳng hạn như nhật ký hệ thống và Giao thức quản lý mạng đơn giản (SNMP).



Hình 3‑1 Sơ đồ hệ thống API gateway mang tính sẵn sàng cao [2]

Kiến trúc thể hiện trong sơ đồ Hình 3‑1 được mô tả như sau:

* Client bên ngoài gửi yêu cầu kết nối qua một giao thức cụ thể (ví dụ: HTTP, JMS hoặc FTP) tới bộ cân bằng tải
* Load balaner kiểm tra sự sẵn sàng (hellth check) trên từng phiên bản gateway và phân phối tải đến gateway đó
* Mỗi phiên bản gateway có một kết nối bên ngoài (external connections) với hệ thống khác bao gồm: cơ sở dữ liệu (mysql), các kho lưu trữ xác thực (CA SiteMinder, Oracle Access Manager, Lightweight Directory Access Protocol (LDAP) servers)
* Bộ nhớ đệm được sao chép giữa mỗi phiên bản API Gateway bằng cách sử dụng hệ thống bộ nhớ đệm phân tán dựa trên Ehcache
* Mỗi phiên bản gateway có giao diện remote host chỉ định các kết nối gửi đi đến hệ thống API phía sau và chúng có thể được phân phối tải đến dựa trên các mức độ ưu tiên được chỉ định cho remote host
* Mỗi phiên bản gateaway chứa một hệ thống Apache ActiveMQ, nó có thể được định cấu hình cho HA. ActiveMQ là một Message-oriented middleware (MOM) (Hệ thống trung gian chuyển tải gói tin) giúp cho việc chuyển nhận message theo cơ chế queue as-Synchronized
* Việc quản lý tải được sử dụng bởi Admin Node Manager, API Gateway Manager, and Policy Studio

## Vai trò Load balancing trong hệ thống

Khi kết nối trực tiếp vào một domain không có Load Balancing rất có thể sẽ không kịp xử lý, tải chậm, thậm chí không kết nối được khi máy chủ gặp sự cố. Trường hợp này xảy ra vì có quá nhiều người cùng lúc truy cập, ứng dụng…nhất là các ứng dụng thương mại điện tử và mạng xã hội. Và đây chính là lúc mà Load Balancing thể hiện vai trò của mình.

* Đảm bảo thời gian Uptime

Khi máy chủ gặp sự cố, lưu lượng truy cập sẽ được tự động chuyển đến máy chủ còn lại. Nhờ đó, trong hầu hết mọi trường hợp, sự cố bất ngờ có thể được phát hiện và xử lý kịp thời, không làm gián đoạn các truy cập của người dùng

* Data center linh hoạt

Khả năng linh hoạt trong việc điều phối giữa các máy chủ cũng là một ưu điểm khác của Load Balancing. Tự động điều phối giữa các máy chủ cũ và mới để xử lý các yêu cầu dịch vụ mà không làm gián đoạn các hoạt động chung của hệ thống

* Bảo mật cho Data center

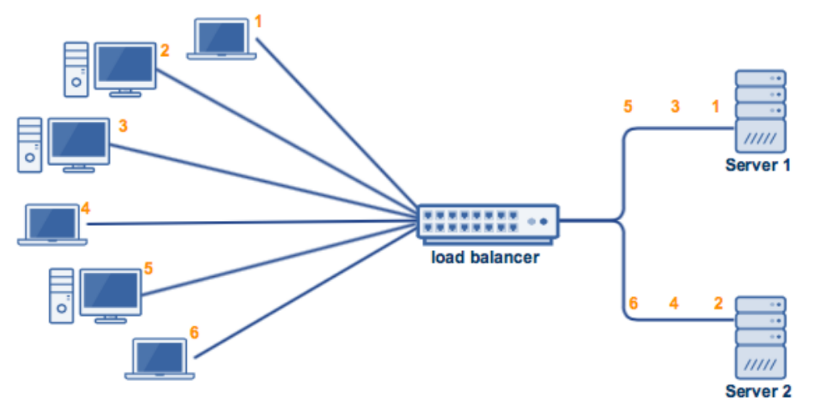
Những yêu cầu từ người dùng sẽ được tiếp nhận và xử lý trước khi phân chia đến các máy chủ. Đồng thời, quá trình phản hồi cũng được thông qua Load Balancing, ngăn cản việc người dùng giao tiếp trực tiếp với máy chủ, ẩn đi thông tin và cấu trúc nội bộ, từ đó chặn đứng những cuộc tấn công mạng hay truy cập trái phép…

Load balancing là một thành phần quan trọng của cơ sở hạ tầng thường được sử dụng để cải thiện hiệu suất và độ tin cậy của các trang web, các ứng dụng, cơ sở dữ liệu và các dịch vụ khác bằng cách phân phối lưu lượng truy cập ứng dụng đến nhiều máy chủ.

## Một số thuật toán cân bằng tải (Load balancing)

### Round Robin

Round Robin là thuật toán lựa chọn các máy chủ theo trình tự. Theo đó, Load Balancer sẽ bắt đầu đi từ máy chủ số 1 trong danh sách của nó ứng với yêu cầu đầu tiên. Tiếp đó, nó sẽ di chuyển dần xuống trong danh sách theo thứ tự và bắt đầu lại ở đầu trang khi đến máy chủ cuối cùng. Đồng nghĩa với các server sẽ nhận lần lượt các request theo thứ tự.



Hình 3‑2 Thuật toán Round Robin

Hình 3‑2 mô tả hệ thống có 6 client được đánh số từ 1 đến 6. Các kết nối 1,3,5 được thực hiến đển server 1 và 2,4,6 được thực hiện đến server 2.

Tuy nhiên, Khi có 2 yêu cầu liên tục từ phía người dùng sẽ có thể được gửi vào 2 server khác nhau. Điều này làm tốn thời gian tạo thêm kết nối với server thứ 2 trong khi đó server thứ nhất cũng có thể trả lời được thông tin mà người dùng đang cần. Để giải quyết điều này, round robin thường được cài đặt cùng với các phương pháp duy trì session như sử dụng cookie. Thuật toán này phù hợp khi các request không có sự sai khác về thời gian xử lý. Xét ví dụ sau:

giây đầu tiên:

* Request 1 -> server 1: 3s xử lý
* Request 2 -> server 2: 1s xử lý

giây thứ 2:

* Request 3 -> server 1: 3s xử lý
* Request 4 -> server 2: 1s xử lý

giây thứ 3:

* Request 5 -> server 1: 3s xử lý
* Request 6 -> server 2: 1s xử lý

Trong trường hợp trên server 1 phải xử lý 3 request mất 9 giây. Trong khi server 2 chỉ mất 3 giây. Như vậy, cân bằng tải chưa tối ưu. Để giải quyết vấn đề này, chúng ta nên sử dụng thuật toán khác: Weighted Round Robin hoặc Least connected

### Weighted Round Robin

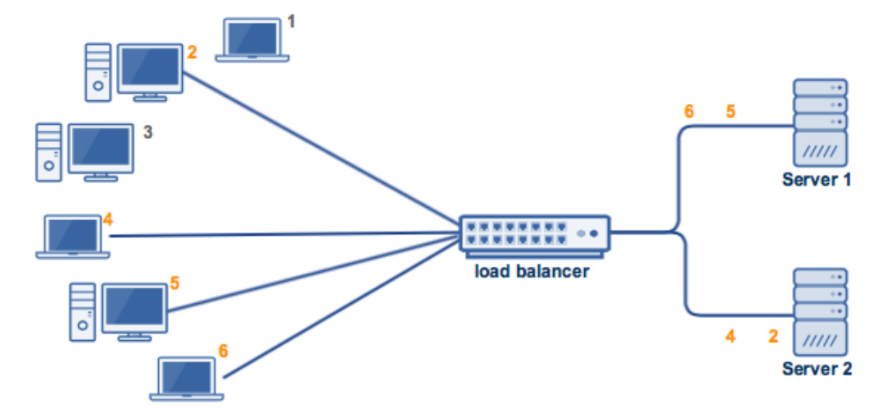
Bản chất giống như thuật toán Round Robin, tuy nhiên chúng ta có thể cấu hình cho một máy chủ nào đó thường xuyên được sử dụng hơn. Mỗi máy chủ được đánh giá bằng một số nguyên. Một server có thông số kỹ thuật gấp đôi server khác sẽ được đánh số lớn hơn và nhận được số request nhiều hơn từ bộ cân bằng tải. Nếu tất cả các máy chủ được đánh trọng số là 1 thì thuật toán này trở thành thuật toán Round Robin.

Với hệ thống như Hình 3‑2 và cấu hình weight 1 = 5 cho server 1, weight 2 =1 cho server 2 thì các kết nối 1,2,3 ,4 5 được thực hiện đến server 1 và kết nối 6 được thực hiện đến server 2.

### Least connected

Quay lại ví dụ của Hình 3‑2

* Có 6 client request đến LB đang cân bằng tải cho 2 server
* Giả sử client 1 và 3 ngắt kết nối trước khi client 6 gửi request, trong khi đó request của client 4,2 chưa được xử lý xong. Việc client 6 được kết nối server 2 có thể làm server 2 trở nên bận rộn, thậm chí bị từ chối vì đầy hàng đợi.
* Giải pháp: request của client 6 sẽ được chuyển đến server 1 (Hình 3‑3)



Hình 3‑3 Thuật toán Least connected

Như vậy, load balancer sẽ phân bố request xuống máy chủ đang xử lý ít request nhất

### Session persistence – Phiên hoạt động

Chẳng hạn chúng có một ứng dụng yêu cầu đăng nhập, nếu khi đăng nhập, session lưu trên **Backend 1**, sau một hồi request lại được chuyển tới **Backend 2**, trạng thái đăng nhập bị mất, hẳn là người dùng sẽ không dung ứng dụng này nữa

Trong các thuật toán trên, mỗi request tiếp theo có thể sẽ được phân phối đến một máy chủ khác nhau. Không có gì đảm bảo rằng cùng một client sẽ luôn được chuyển đến cùng một máy chủ. Vì vậy tạo phiên của client đó làm sao để luôn kết nối tới hoặc liên tục kết nối tới một máy chủ cụ thể - cơ chế cân bằng tải ip\_hash lúc này có thể sử dụng.

Với ip-hash, địa chỉ IP của client được sử dụng như khóa của hàm băm để xác định máy chủ nào trong nhóm máy chủ sẽ được chọn xử lý các request của client. Phương pháp này đảm bảo rằng các yêu cầu từ cùng một client sẽ luôn được chuyển đến cùng một máy chủ trừ khi máy chủ này không khả dụng. Thuật toán này đặc biệt hiệu quả đến các hệ thống cần lưu dữ liệu sau mỗi phiên hoạt động, chẳng hạn giữ lại các mặt hàng trong giỏ hàng giữa các phiên.

### Least Time

Việc quyết định sử dụng server nào phụ thuộc vào lượng request active của server và độ trễ khi xử lý của server. Nó sẽ chọn server ít connection và độ trễ thấp nhất. Tuy nhiên việc tính độ trễ này phụ thuộc vào tham số đầu vào least\_time. Cụ thể như sau:

connect -> Thời gian connect đến upstream server

first\_byte -> Thời gian nhận về những byte dữ liệu đầu tiên

last\_byte -> Thời gian nhận hết toàn bộ dữ liệu

Trên đây là các thuật toán thường được sử dụng phổ biến. Ngoài ra còn một số thuật toán khác mà báo cáo này chưa đề cập.

# PHÁT TRIỂN PHẦN MỀM

## Giới thiệu và cài đặt JHipter

JHipster là một nền tảng phát triển để tạo, phát triển và triển khai các ứng dụng web và kiến ​​trúc Microservice một cách nhanh chóng. Jhipster hỗ trợ nhiều công nghệ frontend, bao gồm: Angular, React, Vue và các ứng dụng di động (Ionic, React Native). Về phía backend, Jhipster hỗ trợ Spring Boot (Java hoặc Kotlin), Micronaut, Quarkus, Node.js, và .NET [3].

Điều kiện tiên quyết để cài đặt JHipster:

* Cài đặt Java
* Cài đặt Git
* Cài đặt Node.js

Sau đó, cài đặt JHipster bằng câu lệnh: npm install -g generator-jhipster.

Tạo project trên Jhipster như sau:

* Tạo 1 thư mục trống là nơi sẽ chứa project: mkdir myapplication
* Chuyển đến thư mục vừa tạo: cd myapplication/
* Generate ứng dụng: jhipster
* Lựa chọn những option phù hợp với project của bạn

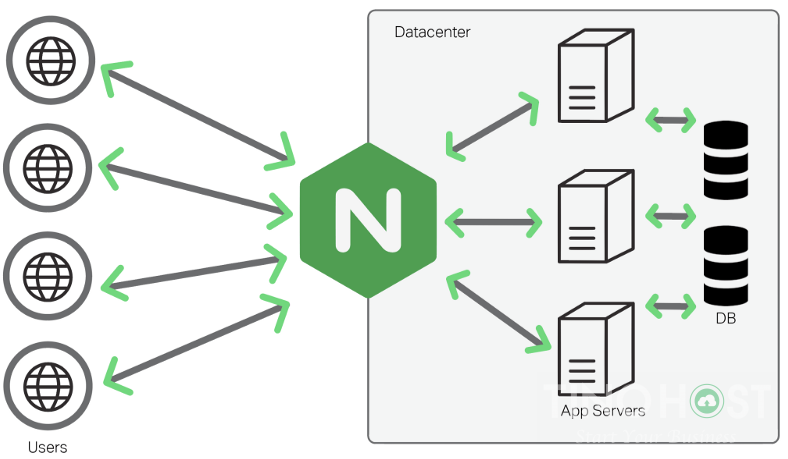
## Giới thiệu và cài đặt Nginx

### Giới thiệu Nginx

[Nginx](https://www.nginx.com/) là một máy chủ proxy ngược mã nguồn mở (open source reverse proxy server) sử dụng phổ biến giao thức HTTP, HTTPS, SMTP, POP3 và IMAP. Nginx thường được chọn để cân bằng tải (load balancer), HTTP cache và máy chủ web (web server). NGINX là một web server mạnh mẽ và sử dụng kiến trúc đơn luồng, hướng sự kiện. Hiện nay, Nginx được ưa chuộng và có mặt trên khắp máy chủ Linux bởi sự hiệu quả của nó, điển hình [Facebook](https://facebook.com/), [Microsoft](https://www.microsoft.com/), Google, Apple…

### Nguyên lý hoạt động Nginx

Nginx không dựa vào luồng (threads) để xử lý các truy vấn (request) mà theo kiến trúc hướng sự kiện (event-driven) không đồng bộ (asynchronous) và có khả năng mở rộng.



Hình 4‑1 Nguyên lý hoạt động Nginx [4]

Với các web server truyền thống, khi một yêu cầu gửi lên để mở một trang web. Trình duyệt sẽ liên lạc với server chứa website đó. Sau đó, server sẽ tìm kiếm đúng file yêu cầu của trang đó và gửi ngược về cho server. Như vậy, tạo một luồng duy nhất cho mọi yêu cầu, nhưng Nginx không hoạt động theo cách đó. Nginx thực hiện với kiến trúc hướng sự kiện không đồng bộ. Điều đó có nghĩa là các luồng tương tự được quản lý theo một worker process và mỗi worker process chứa các đơn vị nhỏ hơn gọi là worker connection. Toàn bộ các đơn vị này sau đó chịu trách nhiệm xử lý các luồng yêu cầu. Worker connection cung cấp các yêu cầu cho worker process, cũng sẽ gửi nó đến master process. Cuối cùng, master process cung cấp kết quả của những yêu cầu đó. Do đó, Nginx có thể xử lý hàng ngàn yêu cầu mà không gặp bất kỳ khó khăn nào so với các máy chủ truyền thống.

### Cài đặt Nginx

* Tải bản Nginx cho window tại <https://nginx.org/en/download.html> và giải nén
* Nginx không cần phải cài đặt. Để thao tác với nginx ta mở command line ở folder chứa file nginx.exe và chạy các lệnh sau:

start nginx # bật nginx

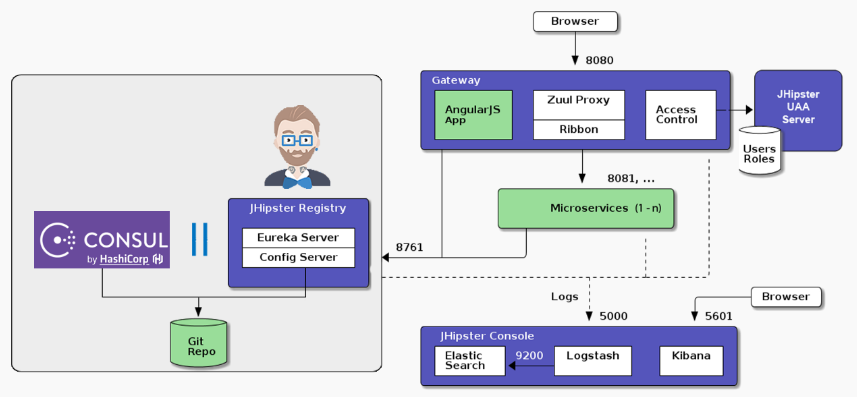
nginx -s stop # tắt nginx

nginx -s reload # tải lại cấu hình nginx (khi file cấu hình thay đổi)

## Phát triển phần mềm trên Jhipster và nginx

### Mô hình hệ thống Microservices trên Jhipster

Sơ đồ hệ thống Microservices JHipster được minh họa như trên Hình 4‑2.



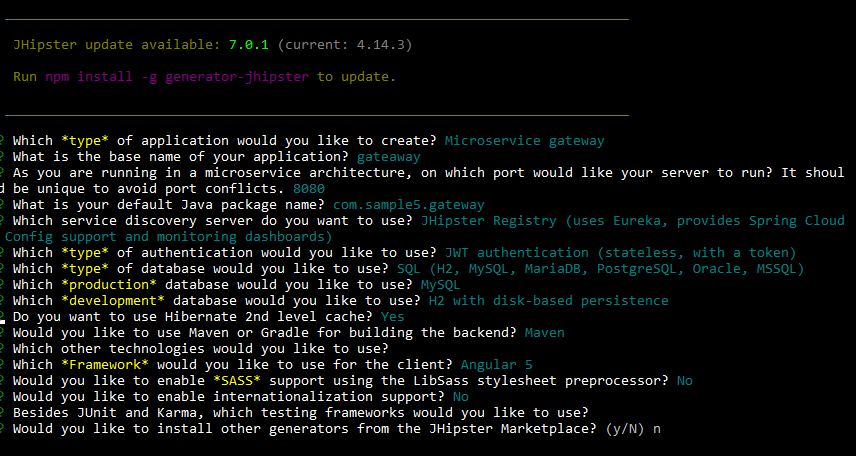
Hình 4‑2 Mô hình hệ thống Microservices trên JHipster [3]

Như sơ đồ kiến trúc trên, ta có thể thấy kiến trúc của hệ thống Microservice có thành phần chính là service registry, các Microservice và Gateway.

* Service registry: là nơi đăng ký và cấu hình tất cả các service bao gồm: số lượng service được triển khai, phiên bản, tình trạng deloy, …
* Microservices: là ứng dụng được tạo ra từ Jhipster thực hiện một nghiệp vụ nào đó như: thanh toán, tài khoản, thông báo và chúng độc lập lẫn nhau
* Gateway: là cổng trung gian giữa lient và service, được dùng để handle các request bằng cách điều hướng chúng đến các service tương ứng và nó cũng có thể được dùng để xác thực, giám sát và làm nhiều việc khác.

### Tạo Gateway trên Jhipster

Trên terminal/cmd, tạo và chuyển đến thư mục gateway và run jhipster. JHipster sẽ nhắc người dùng với nhiều câu hỏi về loại ứng dụng bạn muốn tạo và những tính năng bạn muốn đưa vào. Tạo ứng dụng gateway với các cài đặt như trên Hình 4‑3.



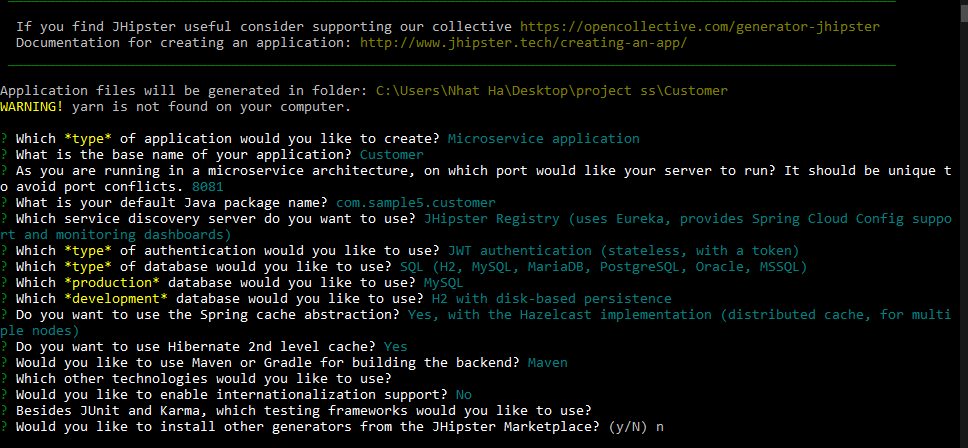
Hình 4‑3 Tạo Gateway trên Jhipster

Trong trường hợp tạo project đơn giản em đã lựa chọn các option như sau:

* Application type: Microservice gateway
* Base name of the application: blog
* Port: 8080
* Default package name: com.sample.gateway
* JHipster Registry: Yes
* Type of authentication: JWT
* Type of database: SQL
* Production database: MySQL
* Development database: H2 with disk-based persistence
* Maven or Gradle: Maven
* Client framework: Angular 5
* Internationalization support: No
* Install other generators from the JHipster Marketplace: No

### Tạo Microservice trêm JHipter

Tương tự với tao Gateway, chọn các tùy chọn sau cho một Microservice như trên Hình 4‑4



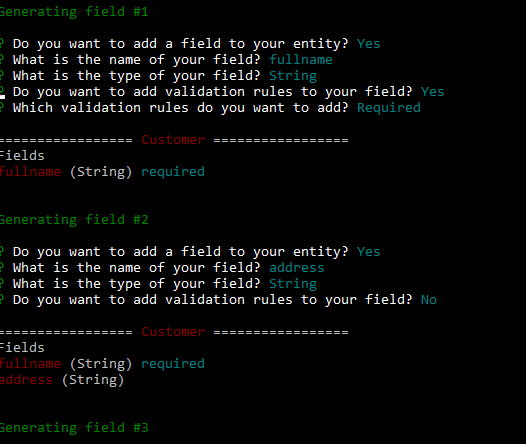
Hình 4‑4 Tạo Microservice trên JHipster

* Type of application: Microservice application
* Application base name: Customer
* Port: 8081
* Default java package name: com.sample5.Category
* Use the JHipster Registry to configure, monitor, as well as scale your Microservices and gateways: Yes
* Type of authentication: JWT authentication (stateless, with a token)
* Type of database: SQL
* Production database: MysqlSQL
* Development database: H2 with disk-based persistence
* Hibernate 2nd level cache: Yes, with HazelCast (distributed cache, for multiple nodes)
* Building tool: Maven
* Other technologies: none
* Internationalization support: No
* JHipster Marketplace: No

### Tạo entity trên JHipter

Để tạo entity gõ lệnh: jhipster entity {tên của entity bạn muốn tạo}

Sau đó jhipster sẽ hiện lên các các câu hỏi để bạn có thể tùy chọn tên các trường, kiểu dữ liệu của các trường, validate và cả kiểu quan hệ với các entity khác nữa.



Hình 4‑5 Tạo entity Customer trên JHipster

Hình 4‑5 là một ví dụ tạo entity Customer trên JHipter. Entity này có các trường: fullname, address, phone và các kiểu dữ liệu, validation rules tương ứng.

### Sử dụng NGINX làm load balancer

Trong khuôn khổ đồ án này em sẽ sử dụng 3 server với vai trò như sau:

• Master (Nginx): Đóng vai trò là Load balancer

• Backend1: Webserver 1

• Backend2: Webserver 2

Mục tiêu của chúng ra là khi client gửi request đến, master server sẽ điều hướng đến một trong 2 backend server, đồng thời không để mất session người dùng. Tiến hành cấu hình Upstream trên Master server như sau:

**http** {

**upstream** backend {

**server** backend1;

**server** backend2;

}

}

Đầu tiên, bạn cần khai báo một nhóm các máy chủ web upstream group. Directive này được đặt trong http block. Ở đây backend1 và backend2 chính là server name của 2 máy chủ web, ta có thể thay bằng địa chỉ IP tương ứng. Để truyền các request từ người dùng vào một group các server, tên của group được truyền vào với directive proxy\_pass (hoặc fastcgi\_pass, memcached\_pass, uwsgi\_pass, scgi\_pass tùy thuộc vào giao thức).

**server** {

**location** / {

**proxy\_pass** http://backend;

}

}

# KẾT QUẢ VÀ KIỂM THỬ

Trong chương này, em xây dựng một kịch bản demo và trình bày kết quả đạt được.

## Kịch bản demo

* Tạo ra 2 bản sao gateway:

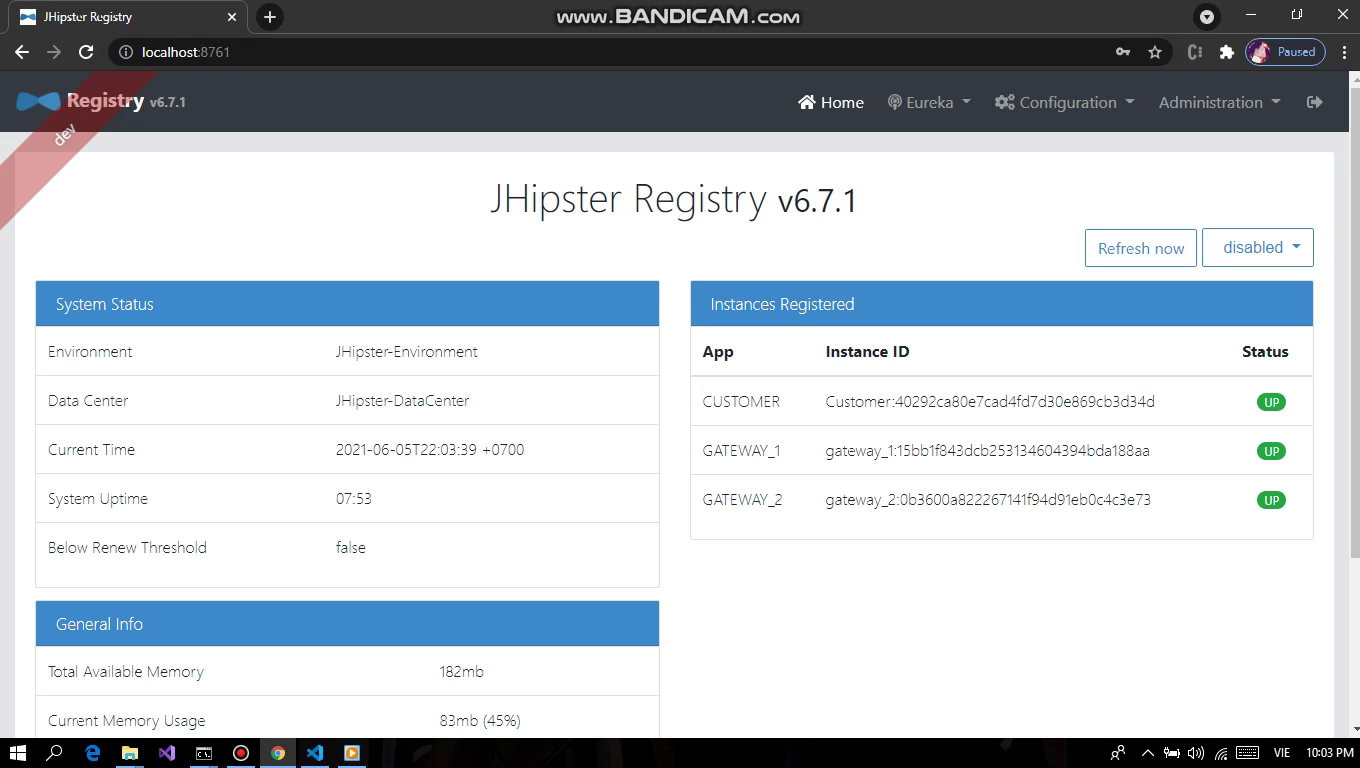
Gateway\_1: được build ở cổng localhost:8080

Gateway\_2: được build ở cổng localhost:8082

* Tạo 1 API micoservice: Customer
* Sử dụng NGINX làm load balancer và áp dụng thuật toán ***Weighted Round Robin.*** Server này được build ở cổng localhost:80. Khởi tạo giá trị weight tại 2 gateway lần lượt là 2 và 1.

## Kết quả đạt được

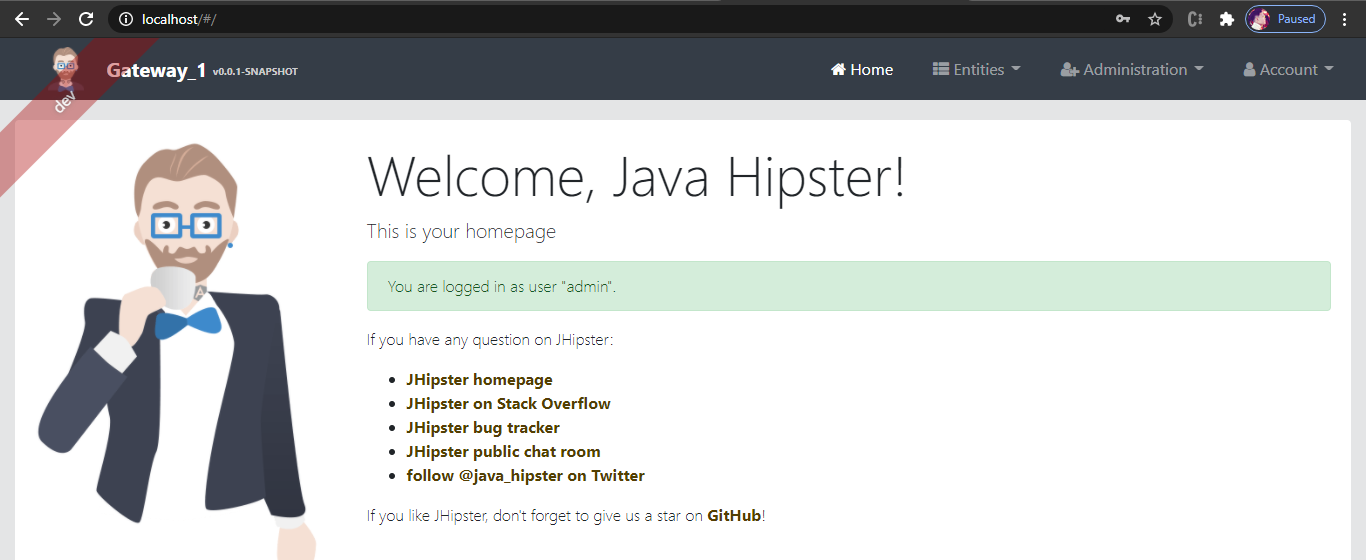
Như trên Hình 5‑1, các instance: Customer, Gateway\_1, Gateway\_2 đã được đăng ký với JHipter Registry.



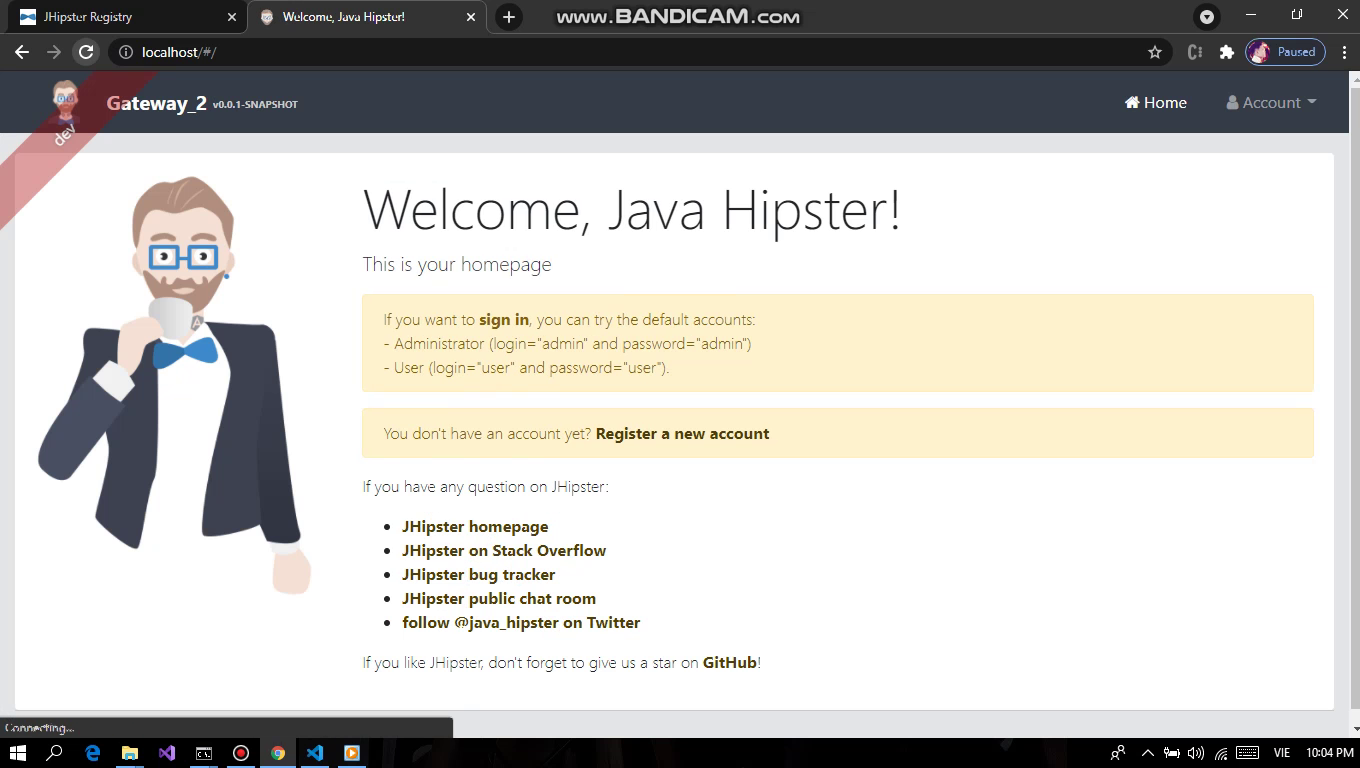
Hình 5‑1 Giao diện quản lý JHipster Registry

Truy cập vào hệ thống bằng cách gửi request đến url: <http://localhost:80/>

Như đã đề cập ở các chương trước, server Nginx sẽ tự điều hướng đến 2 server Gateway\_1 và Gateway\_2 theo một phương pháp cụ thể (quan sát Hình 5‑2, Hình 5‑3)

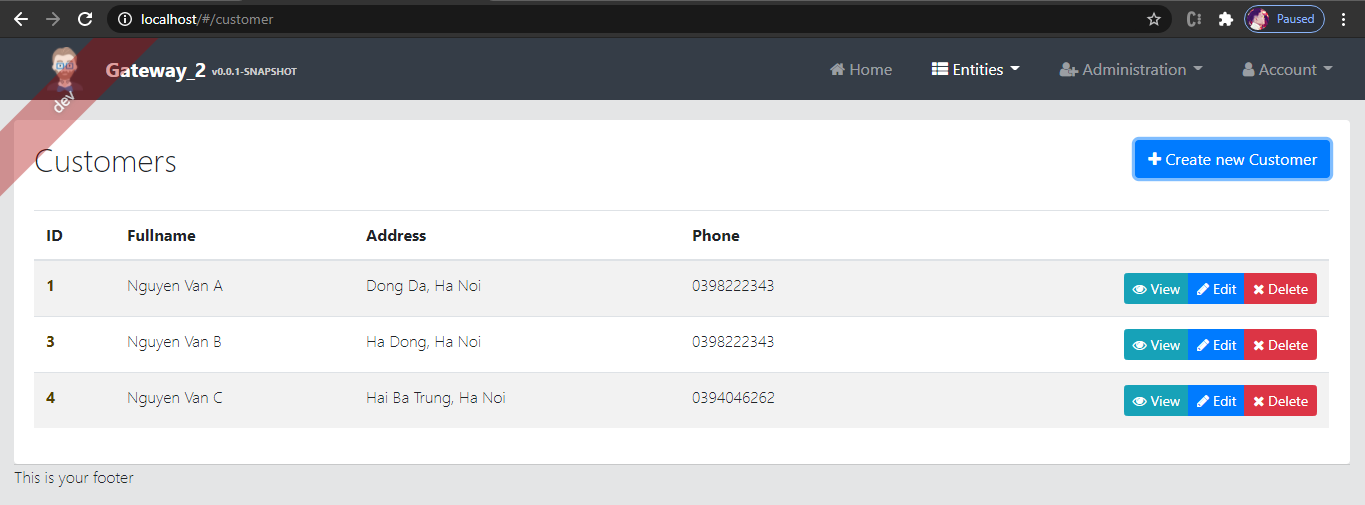


Hình 5‑2 Load balancer điều hướng đến Gateway\_1



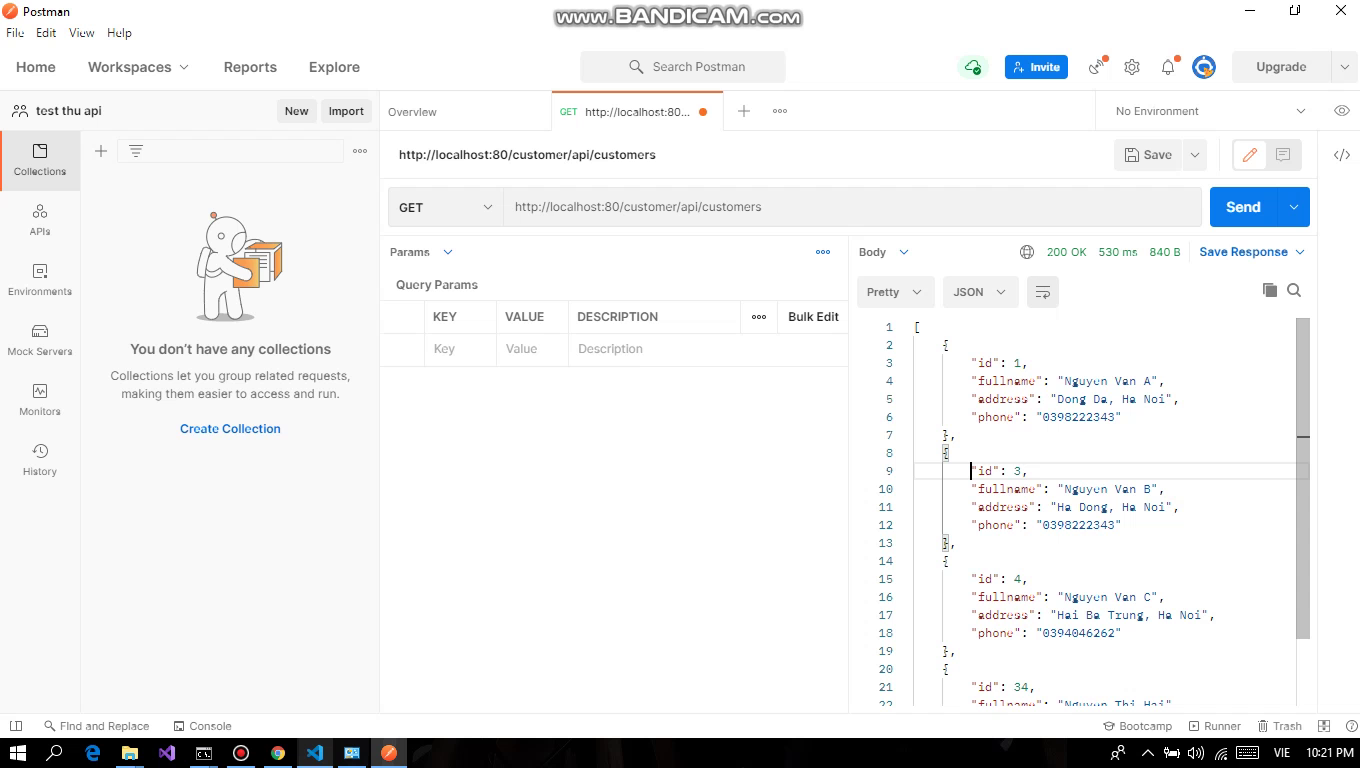
Hình 5‑3 Load balancer điều hướng đến Gateway\_2

Bạn cũng có thể thao tác (xem, sửa, xóa) với các entity bằng cách mở giao diện quản lý entity như trên Hình 5‑4.



Hình 5‑4 Giao diện quản lý entity

Tiến hành gọi BE API qua cổng Nginx (localhost:80) ta nhận được status response: 200 OK. Đông nghĩa với việc request đã được tiếp nhận và xử lý thành công.



Hình 5‑5 gọi BE API qua cổng Nginx

Như vây, em đã xây dựng thành công demo hệ thống gateway api đặt trong chế độ sẵn sàng cao. Qua đó khẳng định tính đúng đắn của bản demo so với lý thuyết được nêu ở các chương trước.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

**Kết luận chung**

Đồ án này trình bày phương pháp giúp tăng tính sẵn sàng cao của API Gateway cũng như các thuật toán load balancing phổ biến.

Thông qua việc demo một hệ thống đơn giản sử dụng công cụ Jhipter, Nginx và áp dụng một phương pháp cụ thể, độ hiệu quả của phương pháp này càng được khẳng định rõ. Kết quả này càng khẳng định thêm tính khả thi khi áp dụng vào các trường hợp thực tế.

Do thời gian còn hạn chế nên việc kịch bản kiểm thử được xây dựng còn đơn giản, chưa đề cập chuyên sâu vào các khía cạnh khác như: các giao thức truyền tải, bộ nhớ đệm, proxy, security…

Em xin đề xuất một số **hướng phát triển** tiếp theo cho hệ thống như sau:

- Xây dựng hệ thống lớn hơn để thấy nhiều vấn đề hơn trong mô hình hệ thống

- Hướng giải quyết khi dữ liệu thành bigdata

- Tối ưu code api: bảo mật, xử lý và trả lỗi…

# DANH MỤC TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | Moesif, [Online]. Available: https://www.moesif.com/blog/technical/api-gateways/How-to-Choose-The-Right-API-Gateway-For-Your-Platform-Comparison-Of-Kong-Tyk-Apigee-And-Alternatives/. [Accessed 3 6 2021]. |
| [2] | "API Gateway 7.6.2 Administrator Guide," Axway, [Online]. Available: https://docs.axway.com/bundle/APIGateway\_762\_AdministratorGuide\_allOS\_en\_HTML5/page/Content/AdminGuideTopics/high\_availability.htm. [Accessed 3 6 2021]. |
| [3] | "Microservices," Jhipster, [Online]. Available: https://www.jhipster.tech/. [Accessed 3 6 2021]. |
| [4] | "Kiến thức Hosting," [Online]. Available: https://blog.tinohost.com/tim-hieu-ve-nginx/. [Accessed 3 6 2021]. |
| [5] | "VIBLO," [Online]. Available: https://viblo.asia/p/api-gateway-la-gi-tai-sao-mot-he-thong-microservices-lai-can-api-gateway-Do754pDX5M6. [Accessed 3 6 2021]. |
| [6] | "Microservices Architecture and Design Patterns," XenonStack, [Online]. Available: https://www.xenonstack.com/insights/microservices/. [Accessed 3 6 2021]. |

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