ĐẠI HỌC QUỐC GIA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN

NGUYỄN TRỌNG PHỔ

NGHIÊN CỨU RESTFUL API VÀ ỨNG DỤNG XÂY DỰNG HỆ THỐNG TOPUP

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

HÀ NỘI - 2016

ĐẠI HỌC QUỐC GIA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN

NGUYỄN TRỌNG PHỔ

NGHIÊN CỨU RESTFUL API VÀ ỨNG DỤNG XÂY DỰNG HỆ THỐNG TOPUP

Ngành: Công nghệ thông tin
Chuyên ngành: Quản lý hệ thống thông tin
Mã số: Chuyên ngành đào tạo thí điểm

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:
PGS.TS. Nguyễn Đình Hóa

HÀ NỘI - 2016

LỜI CAM ĐOAN

Tôi xin cam đoan Luận văn này là công trình nghiên cứu của cá nhân tôi. Các dữ liệu, kết quả nêu trong Luận văn này được xây dựng dựa trên cơ sở nghiên cứu lý thuyết, khảo sát tình hình thực tiễn, dưới sự hướng dẫn của các thầy cô giáo, cùng những đóng góp của các anh chị khóa trên.

Tôi xin cam đoan những điều tôi nói ở trên là đúng sự thật, mọi sai sót tôi xin chịu hoàn toàn trách nhiệm trước Hội đồng.

Tác giả luận văn

Nguyễn Trọng Phổ

LỜI CẢM ƠN

Để hoàn thành tốt Luận văn tốt nghiệp với đề tài ***"Nghiên cứu RESTful API và ứng dụng xây dựng hệ thống TOPUP"*** ngoài sự nỗ lực, cố gắng của bản thân tôi, không thể thiếu sự giúp đỡ, hướng dẫn của các thầy cô. Qua đây, tôi xin gửi lời cảm ơn chân thành đến Thầy giáo ***PGS.TS. Nguyễn Đình Hóa*** đã tận tình hướng dẫn, giúp đỡ tôi có những định hướng đúng và hoàn thành tốt đề tài nghiên cứu của mình.

Tôi cũng xin gửi lời cảm ơn tới gia đình và bạn bè đã luôn quan tâm, động viên, giúp đỡ và tạo điều kiện cho tôi để tôi có điều kiện tốt nhất trong quá trình thực hiện đề tài.

Mặc dù đã có nhiều cố gắng nhưng do hạn chế về thời gian cũng như kiến thức, trình độ cá nhân nên đề tài nghiên cứu của tôi không thể tránh khỏi những thiếu sót. Vì vậy, tôi rất mong nhận được góp ý, chỉ bảo của các thầy cô giáo để đề tài nghiên cứu của tôi được đầy đủ, hoàn thiện hơn và có thể ứng dụng vào thực tế hiệu quả nhất.

Tôi xin chân thành cảm ơn!

Tác giả luận văn

Nguyễn Trọng Phổ

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT

STT	Từ viết tắt	Viết đầy đủ
1	API	Application Programming Interface
2	REST	Representational State Transfer
3	WSDL	Web Service Description Language
4	SOAP	Simple Object Access Protocol
5	HTTP	Hypertext Transfer Protocol
6	XML	EXtensible Markup Language
7	UDDI	Universal Description, Discovery và Integration
8	RPC	Remote Procedure Call
9	URI	Uniform resource identifier
10	JSON	JavaScript Object Notation
11	ICP	Internet Cache Protocol
12	HTCP	Hypertext Caching Protocol
13	TLS	Transport Layer Security
14	JWT	JSON Web Token
15	HMAC	Hashing Message Authentication Codes
16	SHA	Secure Hash Algorithm
17	HTTPS	Hyper Text Transport Protocol Secure
18	NSD	Người sử dụng
19	CSDL	Cơ sở dữ liệu
20	SQL	Structured Query Language

DANH MỤC HÌNH, BẢNG, BIỂU

DANH MỤC HÌNH

PHẦN MỞ ĐẦU

1. Cơ sở khoa học và tính cấp thiết của đề tài

Ngày này hệ thống Internet ngày càng phát triển, phần mềm sử dụng hệ thống internet ngày càng nhiều. Các phần mềm đa dạng dẫn đến có rất nhiều yêu cầu cần được đáp ứng. Một số phần mềm đòi hỏi về lượng thông tin lớn, dữ liệu lớn… nhưng không thể lưu dữ liệu đó tại thiết bị sử dụng, một số loại yêu cầu được cập nhật realtime (theo thời gian thực) để đảm bảo sự đúng đắn của thông tin (chứng khoán, tiền tệ ...), một số phần mềm đòi hỏi xử lý nhanh và mạnh, mà các thiết bị lại không thể thực hiện được do cấu hình không đủ.

Thông thường, để sử dụng các dịch vụ đó thì người dùng cần dùng trình duyệt, truy cập website và thực hiện. Nhưng người dùng chỉ có thể sử dụng các giao diện mà nhà cung cấp đã thiết kết sẵn tuy nhiên chúng không đáp ứng những mong muốn của người dùng. Để giải quyết vấn đề trên chúng ta cần xây dựng một ứng dụng có các tính năng như các dịch vụ đó nhưng giao diện thân thiện hơn. Vì vậy cần phải sử dụng những dịch vụ riêng biệt để tương tác với hệ thống cung cấp các dịch vụ nói trên. Một hệ thống như vậy được gọi là API.

Để giải quyết vấn đề trên tác giả đề xuất luận văn ***"Nghiên cứu RESTful API và ứng dụng xây dựng hệ thống TOPUP"*** nhằm nghiên cứu xây dựng một hệ thống API cung cấp cho khách hàng phương án nạp tiền trực tiếp vào tài khoản thuê bao trả trước, trả sau, tài khoản game, học trực tuyến,… bằng các thao tác đơn giản trên điện thoại, máy tính hoặc các thiết bị khác có kết có kết nối internet, GPRS, Wifi hoặc 3G.

2. Mục tiêu và nhiệm vụ của đề tài

- Hiểu được các nguyên tắc của REST.
- Hiểu được loại dữ liệu được điều khiển bởi Tiến hành cài đặt API RESTful theo phương pháp trên các hệ thống dựa trên nền web.
- Đưa ra được phương pháp xây dựng cách thức truy cập dữ liệu sử dụng API REST.
- Tiến hành cài đặt API RESTful theo phương pháp trên.
- Cho thấy rằng API vừa cài đặt có thể dùng chung cho cả người và máy.

3. Ý nghĩa khoa học của đề tài

- Nghiên cứu các giải pháp xây dựng API, so sánh và đưa ra ưu nhược điểm của các giải pháp qua đó đưa ra giải pháp phù hợp nhất để xây dựng API.
- Áp dụng các kết quả đã nghiên cứu để xây dựng, cài đặt và thử nghiệm hệ thống API TopUp gồm các chức năng: kiểm tra số dư, lấy thông tin về các dịch vụ, thực hiện TopUp, lấy lịch sử giao dịch.

4. Phương pháp nghiên cứu

- Thu thập, phân tích các tài liệu và những thông tin liên quan đề đề tài.
- Tìm hiểu các giải pháp trong việc xây dựng API của một số Website trong và ngoài nước.
- Kết hợp các nghiên cứu đã có trước đây của các tác giả trong và ngoài nước cùng với sự chỉ bảo, góp ý của thầy hướng dẫn để hoàn thành nội dung nghiên cứu.

5. Phạm vi nghiên cứu

- Nghiên cứu một số giải pháp xây dựng API
- Do có những hạn chế nhất định về cơ sở vật chất và điều kiện tiếp cận thực tế với lĩnh vực viễn thông nên việc cài đặt ứng dụng chủ yếu mang tính thử nghiệm.

6. Các kết quả nghiên dự kiến cần đạt được

- Nghiên cứu một số giải pháp xây dựng API, quy trình thực hiện TopUp.
- Cài đặt thử nghiệm chức năng TopUp trực tuyến thông qua môi trường web.

7. Bố cục luận văn

Phần nội dung chính của luận văn sẽ được bố cục thành 4 chương chính sau:

Chương 1: Dịch vụ web và REST

Giới thiệu chung về dịch vụ web, kiến trúc và các thành phần cơ bản của dịch vụ web như XML, SOAP, WSDL và UDDI đồng thời giới thiệu về REST, mô tả về REST và sự phù hợp của nó với nền tảng cơ bản với dịch vụ web, đưa ra lý do tại sao chọn REST để phát triển dịch vụ web và giới thiệu dịch vụ RESTful mà tác giả sẽ phát triển trong luận văn này

Chương 2: Bảo mật với RESTful

Chương này giới thiệu các phương pháp bảo mật cơ bản cũng như cách thực hiện và áp dụng vào hệ thống được tác giả trình bày trong chương này

Chương 3: Khung làm việc Laravel

Giới thiệu về khung làm việc Laravel, là một khung làm việc định nghĩa và hỗ trợ thực thi các dịch vụ RESTful.

Chương 4: Xây dựng và phát triển bộ API TOPUP

Chương này sẽ giới thiệu chi tiết về hệ thống TOPUP, nguyên tắc hoạt động của hệ thống cũng như mục tiêu mà hệ thống cần đạt được, áp dụng các nguyên tắc của REST và sử dụng các thư viện của khung làm việc Laravel để thiết kế các RESTfull API ứng dụng vào hệ thống TOPUP, ngoài ra các lược đồ tuần tự sẽ được thiết kế và chỉ ra trong chương này để ta có thể thấy được RESTfull API phân biệt các môđun khác như thế nào và tương tác thế nào.

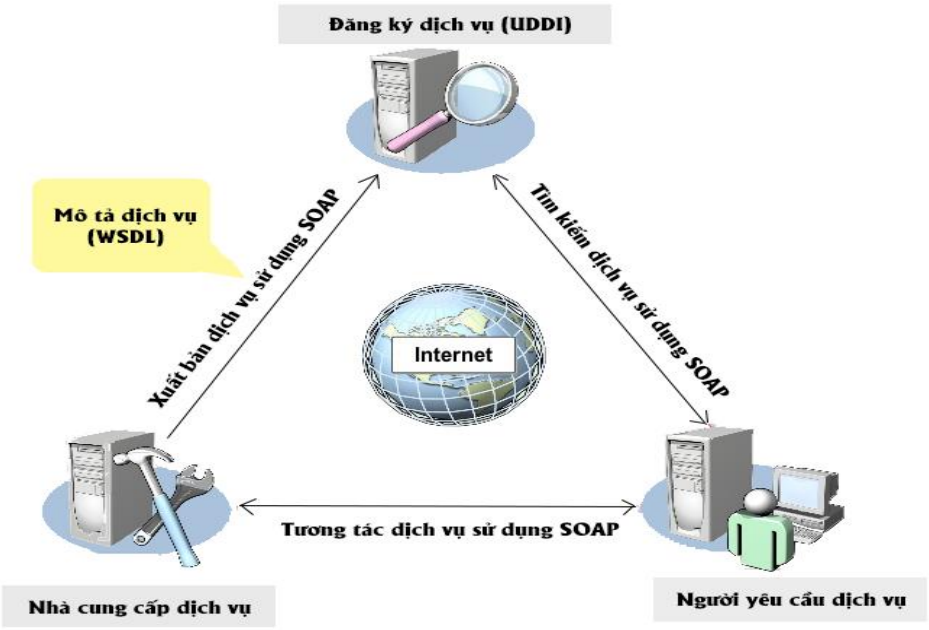
CHƯƠNG 1: DỊCH VỤ WEB VÀ REST

1.1 Tổng quan về dịch vụ web

Theo định nghĩa của W3C (World Wide Web Consortium) [8] thì một dịch vụ web là một hệ thống phần mềm được xây dựng sẵn các tính năng cần thiết, hay còn gọi là các phương thức theo chuẩn để hỗ trợ sự tương tác giữa máy tính và máy tính, giữa người và máy tính. Dịch vụ web cung cấp một API được mô tả theo một định dạng chung gọi là ngôn ngữ mô tả dịch vụ web (Web Service Description Language) viết tắt là WSDL [11]. Người dùng hoặc máy tính thực hiện tương tác với dịch vụ web thông qua giao thức SOAP (Simple Object Access Protocol) [10]. Đây là một trong các giao thức được sử dụng để trao đổi thông tin trên mạng phổ biến nhất hiện nay sử dụng HTTP (Hypertext Transfer Protocol) [2,3] kết hợp với việc sử dụng XML cùng với một số chuẩn khác. Như vậy ta thấy mục đính chính của dịch vụ web là cho phép trao đổi và tương tác thông tin giữa các ứng dụng một cách dễ dàng mà không cần quan tâm đến môi trường phát triển cũng như ngôn ngữ lập trình bởi tất cả đã cả được quy về một định dạng chung. Ngoài ra bản chất của dịch vụ web là một tập hợp các đối tượng, các phương thức được thực thi và công bố lên mạng để có thể triệu gọi được từ xa thông qua các ứng dụng khác nhau.

1.2 Kiến trúc và các thành phần của dịch vụ web

Phần lớn công nghệ dịch vụ web được xây dựng trên mã nguồn mở và được phát triển từ các chuẩn đã được công nhận. Nó tích hợp các ứng dụng trên nền web lại với nhau bằng cách sử dụng các công nghệ XML, SOAP, WSDL, và UDDI [8] trên nền tảng các giao thức Internet với mục tiêu tích hợp ứng dụng và truyền thông điệp. Trong đó XML được sử dụng để đánh dấu dữ liệu, SOAP được dùng để truyền dữ liệu, WSDL được sử dụng để mô tả các dịch vụ có sẵn và UDDI được sử dụng để liệt kê những dịch vụ nào hiện tại đang có sẵn để có thể sử dụng. Chi tiết của các chuẩn mở này chúng ta sẽ bàn chi tiết trong phần sau, phần các thành phần cơ bản của dịch vụ web.



Hình 1.1. Mô tả kiến trúc dịch vụ web

1.2.1 XML

XML (Extensible Markup Langguage) là một chuẩn do W3C đề ra và được phát triển từ SGML, XML là một ngôn ngữ đánh dấu mở rộng với cấu trúc do lập trình viên phát triển dịch vụ tự định nghĩa. Về hình thức thì XML hoàn toàn có cấu trúc thẻ giống như HTML, nhưng HTML định nghĩa thành phần được hiển thị như thế nào thì XML lại định nghĩa những thành phần đó chứa những cái gì. Hay nói cách khác XML có cú pháp tương tự HTML nhưng không tuân theo một đặc tả quy ước như HTML. Người sử dụng hoặc các chương trình có thể quy ước định dạng các thẻ XML. Dịch vụ web là sự kết hợp của nhiều thành phần khác nhau, và dịch vụ này hỗ trợ tương tác giữa các hệ thống được cài đặt trên môi trường khác nhau. Do đó cần phải sử dụng một loại tài liệu đồng nhất giúp giải quyết được vấn đề tương thích và XML hoàn toàn phù hợp với yêu cầu trên. XML đã trở thành nền tảng cho việc xây dựng các dịch vụ web và XML có hai chức năng chính:

- Trao đổi thông tin dữ liệu trong hệ thống sử dụng dịch vụ web
- Mô tả các giao thức sử dụng trong dịch vụ web

1.2.2 SOAP

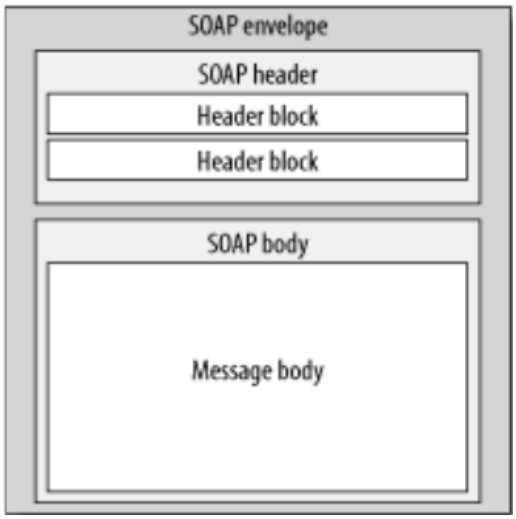
SOAP (Simple Object Access Protocol) là một giao thức dùng để truy xuất các thông tin từ dịch vụ web thông qua một thông điệp chung. SOAP được Microsoft đề xuất vào năm 1998. Hiện nay SOAP thuộc quyền quản lý và cải tiến của tổ chức W3C. SOAP là một giao thức dựa trên nền tảng XML, một giao thức truyền thông hay một định dạng để gửi tin nhắn cho phép các ứng dụng trao đổi thông tin với nhau qua HTTP.

a. Đặc điểm của SOAP

- Khả năng mở rộng (Extensible): Cung cấp khả năng mở rộng phục vụ cho nhu cầu đặc thù của ứng dụng và nhà cung cấp. Các chức năng về bảo mật, tăng độ tin cậy có thể đưa vào phần mở rộng của SOAP. Các nhà cung cấp dịch vụ khác nhau, tùy vào đặc điểm hệ thống của mình có thể định nghĩa thêm các chức năng mở rộng nhằm tăng thêm lợi thế cạnh tranh cũng như cung cấp thêm tiện ích cho người sử dụng.
- Có thể hoạt động tốt trên các giao thức mạng đã được chuẩn hóa (HTTP, SMTP, FTP, TCP,...)
- Có tính độc lập nền, độc lập ngôn ngữ lập trình, mô hình lập trình được sử dụng.

b. Cấu trúc thông điệp của SOAP

Thông điệp SOAP bao gồm phần tử gốc envelope bao trùm toàn bộ nôi dung thông điệp SOAP, và các phần tử header và body. Phần tử header chứa các khối thông tin có liên quan đến cách thức các thông điệp được xử lý như thế nào. Nó bao gồm việc định tuyến và các thiết lập cho việc phân phối các thông điệp. Ngoài ra phần tử Header còn có thể chứa các thông tin về việc thẩm định quyền, xác minh và các ngữ cảnh cho các giao dịch. Các dữ liệu thực sự được lưu trữ tại phần tử body. Bất cứ thứ gì có thể trình bày bằng cú pháp XML đều nằm trong phần tử body của một thông điệp SOAP.



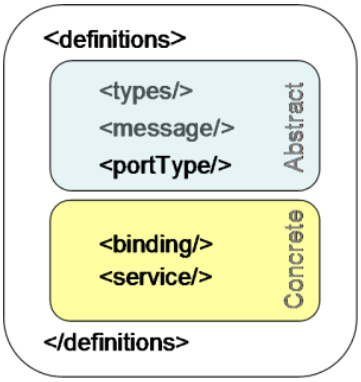
Hình 1.2. Mô tả cấu trúc của một thông điệp SOAP

Tất cả các phần tử envelope đều chứa chính xác một phần tử body. Phần tử body có thể chứa các nốt con theo yêu cầu. Nội dung của phần tử body là các thông điệp. Nếu phần tử envelope mà chứa phần tử header, nó chỉ chứa không nhiều hơn một phần tử header và phần tử header này bắt buộc phải là phần tử con đầu tiên của phần tử envelope. Mỗi một phần tử chứa header đều được gọi là header block. Mục đích của

header block cung cấp giao tiếp các thông tin theo ngữ cảnh có liên quan đến quy trình xử lý các thông điệp SOAP.

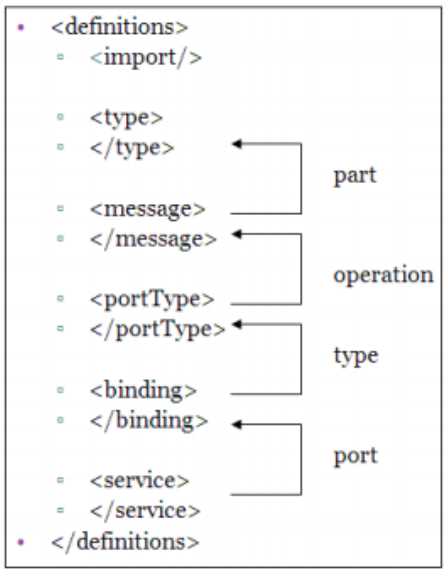
1.2.3 WSDL

WSDL (Web Service Description Language) là một tài liệu đặc tả dựa trên chuẩn ngôn ngữ XML để mô tả các dịch vụ web. Ban đầu WSDL được Microsoft và Ariba đề xuất, nhưng hiện nay WSDL được quản lý và phát triển bởi W3C. Mỗi một đặc tả WSDL sẽ cung cấp tài liệu cho các hệ thống phân tán cũng như mô tả chức năng của một dịch vụ web, cách thức tương tác, các thông điệp tương tác cho các yêu cầu theo request hay response. Sau đây là cấu trúc cơ bản của một tài liệu:



Hình 1.3. Cấu trúc của WSDL

Một đặc tả WSDL bao gồm 2 phần chính: phần trừu tượng (Abstract definitions) và phần cụ thể (Concrete definitions), phần trừu tượng bao gồm các thông tin được chứa trong các thẻ types, message và portypes. Phần cụ thể bao gồm các thông tin được chứa trong các thẻ bindings và ports. Mỗi thành phần sẽ có một tham chiếu đến một thành phần khác được mô tả như hình sau:



Hình 1.4. Các thành phần của WSDL

Mỗi một thành phần có một chức năng riêng, cụ thể như sau:

- Types: chỉ ra kiểu dữ liệu cho các thông điệp gửi và nhận.
- Messages: là một thành phần trừu tượng mô tả cách thức giao tiếp giữa máy khách và máy chủ.
- Porttypes: mô tả ánh xạ giữa các thông điệp, được mô tả trong phần tử messages và các phương thức (operations).
- Binding: xác định giao thức nào được sử dụng khi giao tiếp với dịch vụ web, định nghĩa kiểu binding và giao thức vận chuyển binding cũng định nghĩa các operations.
- Port: chỉ định địa chỉ và cổng kết nối tới dịch vụ web, thường là một địa chỉ URL đơn giản.

1.2.4 UDDI

UDDI (Universal Description, Discovery và Integration) cũng được Microsoft, IBM và Ariba đề xuất năm 2000. Ngày nay UDDI thuộc quyền sở hữu và phát triển của tổ chức OASIS (Organization for the Advancement of Structured Information Standards). UDDI được xây dựng nhằm mục đích cung cấp khả năng cho phép công bố, tổng hợp và tìm kiếm các dịch vụ web. UDDI đưa ra một tập hợp các hàm API được chia làm 2 phần: Inquiry API, dùng để tìm kiếm và truy xuất các dịch vụ web đã đăng ký và Publisher's API, dùng để công bố các dịch vụ web muốn đăng ký. Thông tin tổ chức trong UDDI được chia làm 3 phần:

- White pages: liệt kê thông tin của các nhà cung cấp dịch vụ web, bao gồm địa chỉ, thông tin liên lạc và định danh.
- Yellow pages: phân loại dịch vụ theo tổ chức hay nhóm dịch vụ hoặc địa điểm đặt các dịch vụ.
- Green pages: cung cấp thông tin về các dịch vụ web, cách thức truy cập cũng như tương tác với các dịch vụ web đó.

1.3 XML-PRC

XML như đã nêu trong phần 1.2.1 được viết tắt của cụm từ Extensible Markup Language – Ngôn ngữ đánh dấu dữ liệu. RPC – được viết tắt của cụm từ Remote Procedure Call – Thủ tục gọi từ xa. RPC cung cấp cho người dùng để định nghĩa ra một giao diện mà có thể được gọi từ xa thông qua môi trường mạng máy tính. Giao diện này có thể là một hàm đơn giản nhưng cũng có thể là một thư viện API khổng lồ.

XML – RPC có những đặc điểm sau:

- XML – RPC là một hướng tiếp cận dễ và rõ ràng nhất cho Web Service, nó cung cấp phương thức gọi một ứng dụng từ một máy tính local đến một máy tính từ xa thông qua môi trường mạng.
- XML – RPC cho phép chương trình có khả năng tạo ra các hàm hoặc các thủ tục gọi hàm thông qua mạng máy tính.
- XML – RPC sử dụng giao thức HTTP để vận chuyển thông tin từ Client đến Server.
- XML – RPC sử dụng ngôn ngữ XML để mô tả các thông điệp yêu cầu và các thông điệp đáp ứng gần gũi với ngôn ngữ tự nhiên.
- XML – RPC phía khách chỉ ra cụ thể các thông tin về tên thủ tục, các tham biến trong thông điệp XML yêu cầu, và máy chủ trả về lỗi hoặc trả về thông điệp XML trả lời.
- Các tham số của XML-RPC đơn giản chỉ là kiểu dữ liệu và nội dung – tuy nhiên các cấu trúc dữ liệu phức tạp như struct, array cũng được hỗ trợ bởi XML –RPC.

Sử dụng HTTP có nghĩa là các yêu cầu XML-RPC phải được đồng bộ và phi trạng thái, một yêu cầu XML-RPC luôn luôn có một trả lời XML-RPC tương ứng, bởi vì yêu cầu và trả lời đều phải xảy ra trên cùng một kết nối HTTP.

Phi trạng thái (stateless) có nghĩa là mọi yêu cầu HTTP đều hoàn thành một cách riêng biệt. Khi ở máy khách tạo ra một yêu cầu HTTP thì tất cả các thông tin trong yêu cầu đó phải được đệ trình lên máy chủ. Dịch vụ thường không dựa vào thông tin của yêu cầu trước. Thông điệp XML yêu cầu và XML trả lời là hai thông điệp hoàn toàn riêng biệt. Điều này nhiều khi có thể tránh được các chi phí lớn liên quan đến việc bảo trì hệ thống. XML-RPC không cung cấp hỗ trợ duy trì trạng thái, nhưng với một hệ thống hữu trạng thái thì XML-RPC có thể thực hiện hỗ trợ duy trì trạng thái.

Với các điểm chính về công nghệ liên quan đến XML-RPC như trên thì XML-RPC có các nhược điểm như sau:

- Một yêu cầu XML-RPC bao gồm hành động để thực hiện và các tham số của hành động đó trong yêu cầu gửi lên HTTP khi mà HTTP đã sẵn sàng đáp ứng yêu cầu và trả lời yêu cầu.
- Một API XML-RPC thì cần phải định nghĩa mã lỗi riêng của nó, khi đó sẽ dễ dàng sử dụng trạng thái mã lỗi hơn.
- Với kiểu API sử dụng XML-RPC thì các chức năng về xác thực và cache bên phía máy khách không thể thực hiện được trong hệ thống này.

Một giải pháp mới có thể thay thế XML-RPC để khắc phục các nhược điểm của XML-RPC đó chính là dịch vụ RESTful. Chúng ta sẽ tìm hiểu dịch vụ này chi tiết hơn trong chương 2 để có thể hiểu REST được thay thế XML-RPC như thế nào.

1.4 REST

REST (Representational State Transfer) một kiểu kiến trúc lần đầu tiên giới thiệu vào năm 2000 bởi Roy Fielding [6] trong luận án của ông "Architectural Styles and the Design of Network-based Software Architectures" (Phong cách kiến trúc và thiết kế kiến trúc phần mềm dựa trên mạng) tại Đại học California. Mục đích của REST là thiết kế các ứng dụng mạng phân tán sử dụng HTTP như là một giao thức tầng ứng dụng và nó là một mô hình kiến trúc thực sự cho web.

1.5 Nguyên tắc REST

Sự phát triển ngày càng lớn của các dịch vụ web dẫn tới hệ quả tất yếu là RESTful đã được đưa ra như là một giải pháp để thay thế việc thực hiện triệu gọi từ xa (RPC) thông qua web.

REST là một kiểu kiến trúc cho hệ thống phân tán như World Wide Web. REST được sử dụng rất nhiều trong việc phát triển các ứng dụng Web sử dụng giao thức HTTP trong giao tiếp thông qua mạng internet. Các ứng dụng sử dụng kiến trúc REST này thì sẽ được gọi là ứng dụng phát triển theo kiểu RESTful [6].

Kiến trúc REST cũng phải dựa vào các nguyên tắc như mô tả trong các tài liệu [7,12,13], đó là Tài nguyên (Resources), Khả năng đánh địa chỉ (Addressability), Phi trạng thái (Statelessness), Kết nối (Connectedness), Giao diện đồng nhất (Uniform Interface) và khả năng lưu cache (Cacheability).

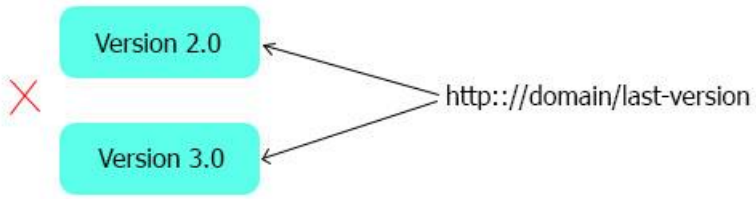
1.5.1

1.5.2 Tài nguyên

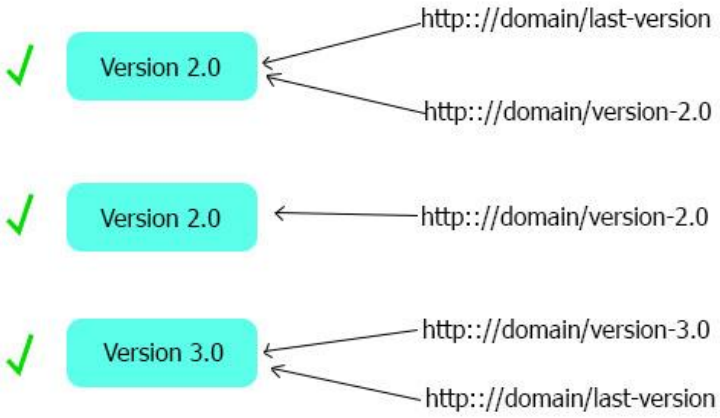
REST tập trung vào việc xử lý các tài nguyên. Tài nguyên là mọi thứ như khách hàng, video, tranh ảnh, trang web… Tài nguyên có thể là một đối tượng vật lý cũng có thể là một khái niệm trừu tượng nào đó. Các tài nguyên này giúp ta định nghĩa được các dịch vụ trong hệ thống, kiểu thông tin mà nó trả về, và hành vi xử lý thông tin của nó. Mỗi tài nguyên đều được định danh bởi một ID duy nhất là URI. Nếu một thông tin nào đó không có URI thì nó không phải là tài nguyên và không tồn tại trên mạng.

Hai tài nguyên không thể có cùng chung một URI (Hình 1.5) nhưng hai URI có thể cùng trỏ vào một tài nguyên vào cùng một thời điểm (Hình 1.6). Ví dụ chúng ta có một URI xác định http://domain/last-version , khi last-version tại phiên bản 2.0 thì cả

hai URI đều cùng trỏ vào một tài nguyên. Sau một thời gian thì last-version lên phiên bản 3.0 lúc đó hai URI này sẽ trỏ vào hai tài nguyên khác nhau.



Hình 1.5. Hai tài nguyên cùng 1 URI



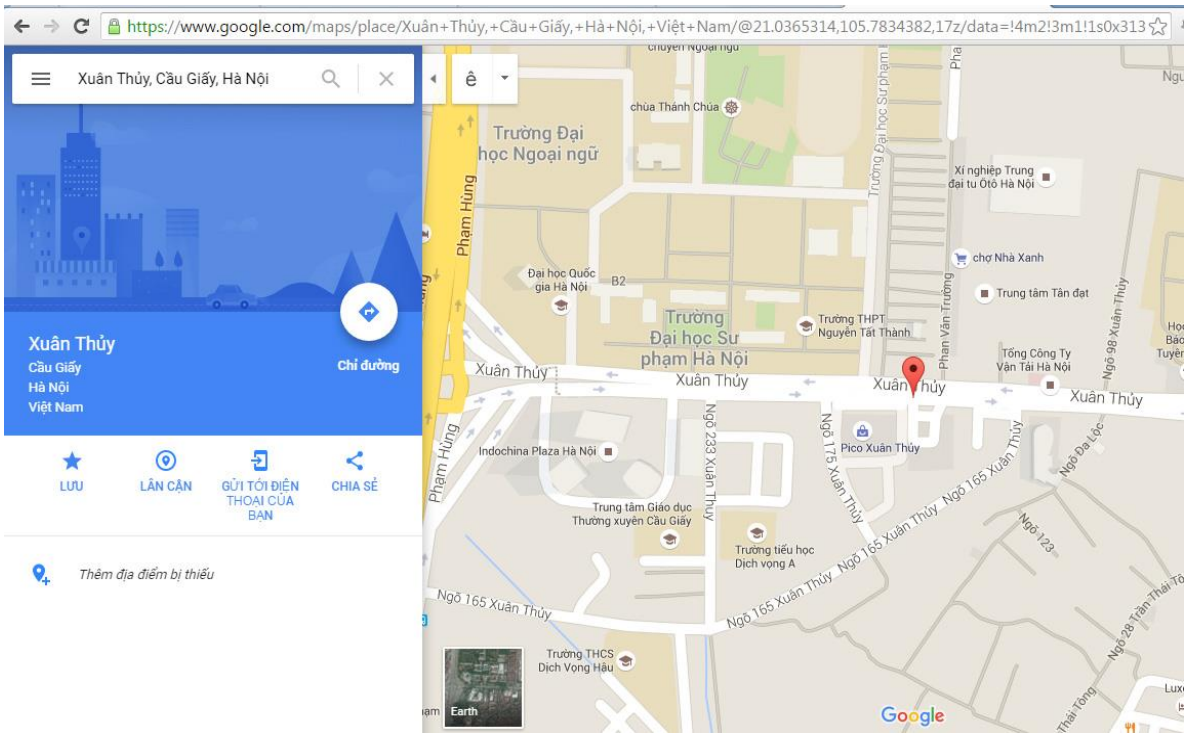
Hình 1.5. Hai URI cùng trỏ đến một tài nguyên

1.5.3 Khả năng đánh địa chỉ

Mọi tài nguyên đều được đánh địa chỉ. Mỗi tài nguyên đều được đánh địa chỉ có nghĩa là mỗi tài nguyên sẽ có một URI. Với khả năng đánh địa chỉ chúng ta có thể lưu lại các thông tin cần thiết, có thể gửi URI này tới người khác như là một tài nguyên, đặc biệt khả năng lưu cache, thì tài nguyên được lưu ở máy khác, sau lần truy cập đầu tiên thì tài nguyên sẽ được truy cập ở máy khách.

Chúng ta xem xét qua ứng dụng Google Maps, vào ứng Google Maps (http://www.google.com/maps) chúng ta gõ "Xuân Thủy, Cầu Giấy, Hà Nội" hệ thống Google Maps sẽ hiển thị ra một địa chỉ liên quan tới từ khóa ta vừa gõ, ta thấy URI của site http:://www.google.com/maps đã thay đổi thành (https://www.google.com/maps/place/Xu%C3%A2n+Th%E1%BB%A7y,+C%E1%BA%A7u+Gi%E1%BA%A5y,+H%C3%A0+N%E1%BB%99i,+Vi%E1%BB%87t+Nam/@21.0365314,105.7834382,17z/data=!3m1!4b1!4m2!3m1!1s0x3135ab358396c7a7:0x8c0029a430be510) có nghĩa rằng Google Maps đã đánh địa chỉ cho kết quả tìm kiếm "Xuân Thủy, Cầu Giấy, Hà Nội". Nếu chúng ta muốn gửi thông tin bản đồ này

cho người khác thì chúng ta chỉ cần gửi URI này thì họ có thể xem được bản đồ mà chúng ta đang xem.



Hình 1.6. Minh họa tìm kiếm bản đồ trên Google Maps

1.5.4 Phi trạng thái

Mọi yêu cầu HTTP đều hoàn thành một cách riêng biệt nhau. Có nghĩa là mỗi một yêu cầu hoàn chỉnh, độc lập không đòi hỏi máy chủ phải thu thập được bất kỳ ngữ cảnh hoặc trạng thái nào của ứng dụng trong lúc xử lý yêu cầu. Nếu máy chủ yêu cầu dữ liệu của yêu cầu trước thì máy khách phải gửi lại thông tin đó xem như là một yêu cầu mới, máy chủ không giữ bất kỳ thông tin gì của máy khách cả.

Điều này làm cho hệ thống đáng tin cậy hơn, đơn giản hơn và khả năng mở rộng lớn hơn. Khi các máy khách gửi yêu cầu đến máy chủ A nhưng vào thời điểm đó máy chủ A lỗi thì một máy chủ khác được thay để xử lý các yêu cầu mà máy khách đã gửi. Điều này có nghĩa là máy chủ web được thay thế một cách dễ dàng và làm cho hệ thống có khả năng thay đổi. Đặc biệt nếu trong hệ thống có sự cân bằng tải thì máy chủ sẽ phục vụ tốt hơn đối với các người dùng mà đã yêu cầu trước đó. Với cân bằng tải này thì hệ thống sẽ trở nên đơn giản hơn để thực hiện.

1.5.5 Kết nối

Dịch vụ kiểu REST cho phép các máy khách chuyển từ trạng thái này đến trạng thái khác bằng cách gửi các liên kết trong các đại diện với nhau, đại diện có thể là siêu âm thanh, có thể là tài liệu mà trong đó không chỉ chứa mỗi dữ liệu mà có thể còn có

cả các liên kết tới các tài nguyên khác nữa. Tất cả các tài nguyên thì nên kết nối và liên kết với nhau, nó cho phép các máy khách biết được nhau bằng cách duyệt các siêu liên kết trong các đại diện tài nguyên.

```
HTTP/1.1 200 OK
Date: ...
Content-Type: application/xml

<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
    <actor>http://example.com/actors/567</actor>
</movie>
```

Hình 1.7. Minh họa đại diện là một liên kết

1.5.6 Giao diện đồng nhất

Máy khách tương tác với hệ thống thông qua các phương thức của HTTP: GET, POST, PUT, DELETE. Các phương thức này định nghĩa được những thứ mà có thể làm với một tài nguyên.

Phương thức GET

Phương thức GET có nghĩa là nhận bất kỳ thông tin gì, trong trường hợp thông tin dữ liệu là dạng form thì được xác định bởi một yêu cầu URI [2], còn trong trường hợp là dịch vụ web RESTful thì thông tin này được xác định bằng một URI là đại diện của một tài nguyên. GET là phương thức an toàn và không thay đổi. An toàn ở đây có nghĩa là nó không làm thay đổi trạng thái của máy chủ, nó chỉ nhận thông tin thôi nên nó không làm ảnh hưởng gì đến máy chủ, không thay đổi ở đây có nghĩa là có thể nhiều yêu cầu giống hệt nhau mà có thể cho kết quả như nhau. Từ khi giao thức HTTP là một giao thức phi trạng thái thì cũng được quy định là an toàn và không đổi. Sử dụng tính chất không đổi cho phép GET lấy lại tài nguyên lặp nếu gặp lỗi.

Nếu yêu cầu thành công và kết quả là một tài nguyên được trả về thì thông điệp trạng thái trả về phù hợp với yêu cầu đó là 200 (thành công). Nếu tài nguyên không tồn tại thì trạng thái trả về là 404 (không thành công, tài nguyên không tìm thấy).

Nếu thông điệp yêu cầu có chứa cả các trường phạm vi, điều kiện thì ngữ nghĩa của phương thức GET có thay đổi một chút. Phương thức này giảm thiểu sự sử dụng mạng không cần thiết bằng cách cho phép nhận một phần thông tin để hoàn thành phương thức GET mà không cần phải chuyển hết dữ liệu mà máy khách đang giữ.

Phương thức POST

Phương thức POST được sử dụng để yêu cầu tạo tài nguyên mới với dữ liệu đính kèm theo yêu cầu. Nếu yêu cầu POST mà không trả về kết quả, tài nguyên có thể được xác định bởi URI, trạng thái trả về nếu thành công là 200 (thành công). Nếu tạo tài nguyên mới thì trạng thái trả về là 201 (tạo tài nguyên mới) và chứa đựng thông tin mô tả trạng thái của yêu cầu và tham chiếu tới tài nguyên mới ở phần header của thông tin.

Phương thức PUT

Phương thức PUT yêu cầu thực thể đính kèm được cung cấp theo cùng yêu cầu URI. Nếu URI tham chiếu tới tài nguyên đã tồn tại thì thực thể kèm theo sẽ được xem xét thay đổi ở trên máy chủ. Nếu tài nguyên đó chưa tồn tại thì có khả năng URI đó yêu cầu một tài nguyên mới, máy chủ sẽ phải tạo một tài nguyên mới với URI đó, nếu tài nguyên mới được tạo thì trạng thái trả về là 201, nếu tài nguyên đã tồn tại tức là thay đổi tài nguyên thì trạng thái trả về là 200. Nếu tài nguyên mới không được tạo, cũng không thay đổi tài nguyên cũ thì một trạng thái mã lỗi sẽ được trả về thông báo lỗi tương ứng.

Phương thức POST và PUT trông có vẻ giống nhau, tuy nhiên chúng khác nhau về mặt ý nghĩa trong các yêu cầu của chúng, trong phương thức PUT thì URI định nghĩa các thực thể đính kèm với yêu cầu (ở máy khách đã chỉ ra ID của tài nguyên trong URI đó). Ngược lại trong phương thức POST trong URI yêu cầu định nghĩa tài nguyên mà sẽ được tạo mới với dữ liệu đính kèm yêu cầu. Hơn nữa PUT là một phương thức không thay đổi (giống như GET) nhưng POST thì ngược lại.

Phương thức DELETE

Phương thức DELETE yêu cầu máy chủ xóa tài nguyên được xác định trong yêu cầu, nó cũng là một phương thức không thay đổi như phương thức GET và PUT. Nếu xóa thành công thì mã trạng thái 200 (thành công) sẽ được trả về, ngược lại nếu không thành công thì một trạng thái mã lỗi tương ứng sẽ được trả về một cách phù hợp.

1.5.7 Khả năng lưu cache

Để vận hành tài nguyên một cách tốt hơn, nhanh hơn và giảm tải máy chủ ở cả hai phương diện thời gian và lưu lượng một cơ chế được sử dụng là cache. Với cơ chế cache, tài nguyên được lưu lại đâu đó trên máy khách với một điều kiện và một khoảng thời gian xác định. Có hai cơ chế cache trong HTTP là thời hạn và xác thực. Theo RFC 2616 [2] thì mục tiêu của cache trong HTTP/1.1 trong hầu hết các trường hợp là loại trừ khả năng yêu cầu kết nối đến máy chủ, giảm thiểu số lượng yêu cầu kết nối, cũng nhưng yêu cầu khả năng gửi lại kết quả trong nhiều trường hợp khác. Điều này làm giảm thiểu băng thông trên máy chủ, nếu các yêu cầu cũ thì thông qua cơ chế

thời hạn, nếu yêu cầu là mới thì thông qua cơ chế xác nhận. Phương thức POST thì không có khả năng cache nếu không có sự điều khiển cache phù hợp, phương thức GET được thiết kế cho phép cơ chế cache.

Thời hạn

Thời hạn là số thời gian máy chủ thông báo cho tài nguyên được yêu cầu bao nhiêu lâu thì được cache lại trên máy khách.

Thời hạn ở header

Thời hạn ở header cho biết số thời gian tài nguyên hết hạn, việc này yêu cầu sự đồng bộ thời gian giữa máy chủ và máy khách.

Điều khiển cache ở header

Trong HTTP/1.1 có thể thay thế thời hạn ở header bằng cách điều khiển cache ở header, để giải quyết vấn đề đồng bộ của máy chủ và máy khách, làm cho khả năng cache linh hoạt hơn. Điều khiển cache ở header có một tập các mệnh đề điều kiện, được dùng để điều khiển máy khách cache tài nguyên như thế nào [14].

Xác nhận

Xác nhận cho phép máy khách yêu cầu máy chủ xem phiên bản cache của tài nguyên, xác nhận rằng tài nguyên còn dùng được nữa hay không.

1.6 Tại sao lựa chọn REST

Với các nguyên tắc và các lợi ích đã trình bày ở trên mà REST có thể đưa lại thì có thể đưa ra một số lý do chọn REST như sau:

Thứ nhất, mỗi đối tượng vật lý hoặc khái niệm trừu tượng đều có thể xem như một tài nguyên duy nhất, điều này có nghĩa rằng chúng sẽ được truy cập nhanh chỉ với một yêu cầu duy nhất ngay sau khi chúng được xác định, với điều này thì cũng tạo dễ dàng cho người dùng và giảm tải cho máy chủ nếu như tài nguyên chỉ truy cập với một yêu cầu duy nhất.

Thứ hai, với nguyên tắc phi trạng thái thì REST làm cho khả năng linh hoạt của hệ thống tốt hơn khi mà máy chủ không nhận được thông tin của máy khách, điều này có nghĩa rằng các máy chủ được kết nối với hệ thống đều có thể nhận bất kỳ yêu cầu nào từ máy khách và trả lời máy khách một cách phù hợp. Trong trường hợp, máy khách đang sử dụng hệ thống mà máy chủ bị mất kết nối thì máy chủ cũng không phải thông báo cho máy khách và xem đó là lỗi của hệ thống.

Thứ ba, nếu dịch vụ không theo kiểu REST thì rất khó để có thể liên kết với nhau vì các tài nguyên không có khả năng đánh địa chỉ, chúng không kết nối được với nhau, các phương thức của HTTP sử dụng trong mỗi yêu cầu không theo một mẫu chuẩn.

Theo kiểu kiến trúc REST thì máy khách có thể đoán được giao diện tương tác được thiết kế như thế nào và kết nối trực tiếp với thông tin cần thiết sử dụng URI chỉ định hoặc điều hướng qua đại diện sử dụng các liên kết.

Thứ tư, khả năng lưu cache có thể dễ dàng sử dụng, làm giảm tải cho máy chủ và cải thiện thời gian cho máy khách.

Với các lợi ích như trên thì REST là một kiểu kiến trúc thú vị và hấp dẫn, rất đáng để sử dụng vì nó có thể cải thiện dịch vụ theo nhiều cách có thể.

1.7 Dịch vụ web kiểu REST

Khi một dịch vụ web áp dụng các nguyên tắc của REST để thiết kế và phát triển ta gọi nó là dịch vụ kiểu REST (RESTfull). REST là kiểu kiến trúc tầng dưới của web, bởi vậy áp dụng các nguyên tắc REST có nghĩa là sẽ tích hợp trực tiếp vào web hơn là tập trung vào các chức năng. Dịch vụ kiểu REST sử dụng tài nguyên web như là các khái niệm trừu tượng chính [4,6].

Dịch vụ kiểu REST có rất nhiều lợi thế mà web đã đưa ra, các lợi thế có thể kể ra như sau:

- Các kiểu dữ liệu chuẩn và phổ biến được sử dụng để mô tả dữ liệu, các loại dữ liệu này cũng được dùng để mô tả tài nguyên ở phía máy khách tùy theo yêu cầu của máy khách, ví dụ kết quả dữ liệu thống kê thì có thể được cung cấp trong HTML cho người dùng có thể đọc được, trong khi đó ở một số máy khách thì có thể áp dụng vào excel để tính toán dữ liệu.
- Giao diện đồng nhất được cung cấp vì phương thức HTTP chuẩn được sử dụng
- Thực sự độc lập ngôn ngữ.
- Khi sử dụng HTTP thì vượt qua tường lửa không phải là vấn đề.
- Lưu cache có thể làm tăng khả năng thực hiện.
- HTTP được sử dụng trong tầng ứng dụng, bởi vậy tất cả các tính năng chuẩn của HTTP được kết thừa vào trong dịch vụ kiểu REST, các tính năng quan trọng như mã hóa, xác thực và cache.

Các vấn đề thuận lợi này được cung cấp phổ biến trong dịch vụ web 2.0 để phát triển dịch vụ kiểu REST như Yahoo, Amazon, Flickr.

CHƯƠNG 2: BẢO MẬT VỚI DỊCH VỤ WEB KIỂU REST

2.1 Giới thiệu

Với sự đơn giản và dễ sử dụng dịch vụ web theo kiến trúc REST đang trở nên phổ biến trong vài năm qua như là một mô hình thiết kế dịch vụ chiếm ưu thế. Dịch vụ sử dụng một cách rõ ràng các phương thức của HTTP như GET, POST, PUT và DELETE, kết quả trả về phụ thuộc vào ứng dụng, có thể là XML, JSON, TEXT hoặc định dạng khác.

Phương pháp bảo mật phổ biến hiện tại cho người dùng bao gồm xác thực người dùng và bảo mật tầng vận chuyển bằng việc bảo mật phiên làm việc qua mạng, đây là cơ chế bảo mật các URL qua mạng bằng cách xác nhận với các máy chủ, tuy nhiên nhược điểm của cơ chế này bởi việc dựa vào sự hiểu biết của người dùng về liên kết và xác nhận của người dùng, cũng dựa vào nhược điểm này mà vấn đề lừa đảo qua mạng đã xảy ra. Cookies của web được sử dụng để truyền thông tin và thực hiện các thông tin bảo mật trong yêu cầu HTTP trong suốt phiên làm việc. Dữ liệu cá nhân thường được lưu trữ hoặc truyền qua mạng mà không mã hóa, hơn nữa người dùng khó khăn trong việc kiểm soát các thông tin cá nhân truyền qua các dịch vụ, khi đó kẻ xấu có thể truy cập thông tin cá nhân của người dùng và lan truyền nó qua mạng dẫn đến thông tin đó khó có thể kiểm soát tốt được. Cách bảo vệ duy nhất có thể là bảo mật URL, hoặc có thể truyền các thông tin đã được mã hóa, đề phòng trường hợp thông tin bị đánh cắp bởi người khác, nhằm bảo vệ các thông tin cá nhân.

Tuy nhiên cơ chế bảo mật ngày nay không phù hợp với kiến trúc REST trong trường hợp các phiên làm việc bảo mật tạo ra các phiên làm việc với các khóa xác định nhưng nhiều dữ liệu tĩnh được lưu trữ trong bộ đệm web không được bảo vệ và lấy dữ liệu đồng thời từ bộ đệm web được. Điều này làm giảm khả năng mở rộng của kiến trúc REST cho các ứng dụng và dịch vụ yêu cầu điều khiển truy cập tới dữ liệu.

2.2 Kiểu kiến trúc REST phù hợp với bộ đệm web

Phiên bản HTTP 1.1 có bốn phương thức chính cho các yêu cầu từ máy trạm được đặt tên là GET, PUT, POST và DELETE (ngoài ra các phương thức khác như CONNECT hoặc TRADE tuy nhiên chúng ta không bàn luận chúng ở đây).

Lưu trữ nội dung của web cần gắn liền với các URI, các URI này được gắn với các dữ liệu là một phần quan trọng của kiểu REST và ứng dụng với phương thức HTTP GET. Dữ liệu được chuyển đến người dùng thông qua trình duyệt web và được lấy từ máy chủ web qua phương thức HTTP GET. Giữa máy trạm và máy chủ có thể có web proxy và bộ đệm web chứa các yêu cầu URI được hiển thị trong yêu cầu GET. Bộ đệm làm giảm băng thông sử dụng, và đặc biệt là giảm tải cho máy chủ.

Có nhiều bộ đệm web được thực thi trong trình duyệt web và chính trong máy trạm của chúng, như bộ đệm proxy (cái này yêu cầu cấu hình trong trình duyệt web), tất nhiên cũng có giao thức để quản lý nội dung của bộ đệm web như ICP (Internet Cache Protocol), HTCP (Hypertext Caching Protocol). Hơn nữa bộ đệm web có thể kết hợp làm việc với nhau, điều này rất quan trọng khi khả năng mở rộng của dữ liệu phụ thuộc vào các dịch vụ. Giao thức HTTP có cơ chế để có thể điều khiển được bộ đệm cho phép bộ đệm cung cấp kết quả đến máy trạm mà không cần kiểm tra lại thông tin trên máy chủ. Cơ chế xác nhận được sử dụng trong bộ đệm để kiểm tra thời gian quá hạn từ máy của của bộ đệm. Bộ đệm web không hợp lệ với mô hình kiến trúc kiểu REST xảy ra với các phương thức HTTP PUT/POST/DELETE được ứng dụng cho các URI tương ứng. Đối với những yêu cầu thay đổi URI tương ứng bộ đệm có thể không thay đổi phiên bản tương ứng trả lại kết quả cho khách hàng nhưng để cho máy chủ xử lý các hoạt động và cung cấp kết quả trả lại. Nói một cách khác nếu HTTP GET được thiết kế như là một phương thức triệu gọi từ xa cho việc ghi dữ liệu thì bộ nhớ đệm sẽ xem như là đã có một kết quả trong đó cho URI trả lại kết quả cũ. Điều này hoàn toàn đúng cho ứng dụng và dịch vụ logic. Kiểu kiến trúc REST thực thi ứng dụng web đưa ra sự hướng dẫn tốt cho người thiết kế và phát triển. Nó được hiểu như sự tự nhiên của web tuy nhiên không lạm dụng chúng. Nó cũng khuyến khích việc sự dụng kịch bản cho tất cả phiên người dùng xử lý dữ liệu cụ thể như là nội dung dựa vào kết quả theo kịch bản không lưu trữ đệm.

2.3 Khóa mã nội dung đối xứng

Ứng dụng web trong kiểu kiến trúc REST tạo ra các URI theo cách mà các video hoặc hình ảnh có thể dễ dàng được lưu bộ đệm vì lý do mở rộng. Nhưng với công nghệ web hiện tại nếu chúng ta lưu bộ đệm, bất kỳ ai cũng có thể truy cập được dữ liệu nếu biết cụ thể URI của dữ liệu đó, khi mà URI đó không được bảo mật. Vấn đề là nếu ta áp dụng các phiên làm việc bảo mật với TLS và chuyển tất cả các dữ liệu vào một kênh bảo mật riêng thì vấn đề bộ nhớ đệm xem như không còn hiệu quả. Ngoài ra dữ liệu được mã hóa nhiều lần cho mỗi lần máy trạm truy cập dữ liệu, sinh ra dư thừa mã hóa. Nhưng bù lại thì dữ liệu được an toàn và phiên làm việc của người dùng được bảo mật. Một giải pháp có thể để cải thiện vấn đề này là vô hiệu hóa bộ nhớ đệm chung và áp dụng các ứng dụng xác định bộ nhớ đệm gần máy chủ nơi các phiên làm việc bảo mật kết thúc. Tuy nhiên đây cũng không phải là một giải pháp tốt khi nó yêu cầu nền ứng dụng xác định bộ nhớ đệm và không sử dụng các lợi ích của bộ nhớ đệm proxy. Nó cũng yêu cầu máy chủ mã hóa dữ liệu khi truyền qua các đường truyền riêng, điều này cũng sinh ra dư thừa mã hóa, yêu cầu bảo mật dữ liệu trong máy chủ sau khi kiểm soát truy cập.

Hình ảnh là loại dữ liệu mà trong cộng đồng người sử dụng có thể chia sẻ nhiều nhất, tuy nhiên là chia sẻ cho những đối tượng đáng tin cậy, vì vậy hình ảnh cần phải được bảo mật. Ngoài ra, các trang web thương mại yêu cầu người dùng phải trả phí cho nội dung mà họ muốn xem và hạn chế nội dung tới khách hàng mà đã trả, tuy nhiên cùng thời điểm họ muốn kiến trúc của họ mở rộng nhất có thể và cho phép lượng người dùng lớn nhất có thể. Vì vậy có một chút mâu thuẫn giữa hai mục tiêu sử dụng bộ nhớ đệm và mã hóa nội dung.

Vấn đề này sẽ được giải quyết một cách đơn giản, chúng ta sẽ tạo một khóa bảo mật nội dung. Tùy chọn gắn thời gian sống của khóa đó với thời gian của bộ nhớ đệm của URI và cung cấp khóa này cho các khách hàng được phép truy cập nội dung. Tất cả dữ liệu được mã hóa với khóa bảo mật nội dung, sau đó được lưu giữ phía ngoài dịch vụ. Chúng ta yêu cầu xác thực người dùng và quyết định ủy quyền truy cập nội dung (chẳng hạn như thông qua TLS hoặc chứng thực HTTP và cơ chế cấp quyền). Dữ liệu thực tế sau đó được truy cập mà không được bảo mật. Trong mô hình phiên làm việc bảo mật thực tế thì các kênh điều khiển nơi mà khóa bảo mật dữ liệu cung cấp cho máy khách sau khi được xác thực và cấp quyền thích hợp. Kiểu kiến trúc REST cho phép máy khách lưu các URI nội dung dữ liệu được mã hóa mà không cần điều khiển truy cập. Do đó dữ liệu không thể sử dụng cho máy khách nếu không có khóa để giải mã nội dung. Lưu ý rằng kết quả cuối cùng của mô hình này tương tự như mô hình bảo mật web hiện tại với TLS.

2.4 Bàn về giải pháp

Có nhiều lợi thế trong phương pháp bảo vệ nội dung dựa vào các khóa bảo mật nội dung riêng cung cấp cho máy khách thông qua phiên làm việc được xác thực và bảo mật. Lợi ích đầu tiên và quan trọng nhất đó là theo cách này, kiến trúc bảo mật web có thể liên kết tốt hơn với kiểu kiến trúc REST. Điều này mang lại tất cả lợi thế về khả năng mở rộng bộ nhớ đệm của kiểu kiến trúc REST.

Mô hình này giảm tải một cách đáng kể cho máy chủ, nội dung được bảo mật. Khi sử dụng khóa bảo mật nội dung, không cần thiết phải mã hóa cùng một nội dung qua nhiều phiên làm việc. Máy chủ cần phải có cơ chế lưu nội dung một lần duy nhất trong suốt thời gian khả dụng ở bộ đệm, đối với những dữ liệu mà rất lâu mới thay đổi nội dung như video hoặc hình ảnh.

Máy khách có thể lấy nội dung từ cục bộ hoặc từ bộ đệm từ xa, thậm chí không cần đăng nhập vào ứng dụng web một khi đã nhận được khóa bảo mật nội dung. Việc này cũng có hiệu quả đối với các trường hợp sử dụng ngoại tuyến cho ứng dụng web mà yêu cầu bảo mật nội dung và quyền truy cập. Mặt khác mô hình này cũng cho phép nội dung được phép phân phối trước tới máy khách rồi nhận quyền truy cập và khóa

bảo mật nội dung sau. Điều này có khả năng cải thiện trải nghiệm người dùng trên một số dịch vụ, nơi mà nội dung có thể được tải về và hiển thị cho người dùng ngay tức thì khi có khóa giải mã.

Có nhiều lựa chọn thay thế thực hiện. Ban đầu nếu người phát triển ứng dụng web muốn sử dụng mô hình nội dung mã hóa này để mã hóa, giải mã, quản lý khóa có thể thực hiện điều này ở tầng ứng dụng. Một phương pháp tiếp cận khác là sự mở rộng cần thiết chuẩn hóa với giao thức HTTP. Có rất nhiều công việc phải thực hiện để định nghĩa mô hình này làm việc như thế nào, thử nghiệm và thực tế xác định khác nhau như thế nào giữa các hành vi công nghệ.

2.5 Kết luận

Chúng ta đã phân tích mô hình kiến trúc bảo mật web trong phạm vi của REST và kết luận rằng cần đưa ra phương pháp bảo mật phù hợp với nó, và một giải pháp đã được đưa ra đó là sử dụng khóa bảo mật nội dung trong đó chia sẻ thời gian sống của nội dung bộ nhớ và chuyển tới máy khách qua HTTP bảo mật sau khi người dùng được xác thực thích hợp. Chúng ta đã phân tích và đưa ra một số giải pháp về chủ đề để nghiên cứu vấn đề này.

Giải pháp đưa ra có vẻ như cải thiện được khả năng mở rộng ứng dụng web trong trường hợp mà điều khiển truy cập và bảo mật nội dung là cần thiết, áp dụng cho tất cả các loại nội dung. Đây là một vấn đề rất quan trọng vì có nhiều nhà cung cấp mạng lưới nội dung và hàng trăm triệu tập tin cần xác thực quyền truy cập. Cũng không dễ dàng nhận thấy công việc của bộ nhớ đệm như thế nào trong thực tế vì còn có nhiều tùy chọn cấu hình.

Có rất nhiều giải pháp để thực hiện bảo mật với dịch vụ web kiểu REST, tuy nhiên ở đây trình bày hai phương pháp bảo mật phổ biến nhất.

2.5.1 Bảo mật với JSON Web Token

JSON Web Token (JWT) là một chuẩn mở (RFC 7519) định nghĩa một cách cô đọng (compact) và tự chứa (self-contained) để truyền thông tin an toàn giữa các bên như một đối tượng JSON. Thông tin này được xác minh và đáng tin cậy bởi chữ ký số. JWT được ký bằng cách bảo mật (với các thuật toán HMAC) hoặc một cặp khóa công khai và khóa riêng sử dụng RSA.

Một số khái niệm trong định nghĩa:

Cô đọng (Compact): Bởi vì có kích thước nhỏ gọn, JWTs có thể được gửi thông qua một URL, tham số POST hoặc bên trong tiêu đề HTTP. Ngoài ra, kích thước nhỏ giúp cho việc truyền tải thông tin nhanh.

Tự chứa (Self-contained): Phần tải dữ liệu chứa tất cả thông tin cần thiết về người dùng, tránh việc phải truy vấn cơ sở dữ liệu nhiều hơn một lần.

Một trong những tình huống ứng dụng JWT thường gặp, đó là:

Xác thực (Authentication): Tình huống thường gặp nhất, khi NSD đăng nhập, mỗi yêu cầu tiếp theo đều kèm theo chuỗi token JWT, cho phép người dùng có thể truy cập đường dẫn, dịch vụ và tài nguyên được phép ứng với token đó. Đăng nhập một lần (Single Sign On) cũng là một chức năng có sử dụng JWT một cách rộng rãi, bởi vì chuỗi JWT có kích thước đủ nhỏ để đính kèm trong yêu cầu và sử dụng ở nhiều hệ thống khác nhau.

Trao đổi thông tin (Information Exchange): JSON Web Token cũng là một cách hữu hiệu và bảo mật để trao đổi thông tin giữa nhiều ứng dụng, bởi vì JWT phải được ký (bằng cặp khóa công khai và khóa riêng). Ngoài ra, chữ ký cũng được tính toán dựa trên nội dung của header và nội dung payload, nhờ đó, bạn có thể xác thực được nội dung là nguyên bản, chưa được chỉnh sửa hoặc can thiệp. Tuy nhiên, một lưu ý hết sức quan trọng là do cấu trúc của JWT đơn giản nên JWT có thể dễ dàng bị giải mã, do vậy, không nên dùng JWT để truyền các thông tin nhạy cảm.

Cấu trúc của JSON Web Token:

JWT bao gồm ba phần phân cách bởi dấu chấm (.) gồm:

- Header
- Payload (Phần tải dữ liệu)
- Signature (Chữ ký số)

Header

Header bao gồm hai phần chính: loại token (mặc định là JWT - Thông tin này cho biết đây là một Token JWT) và thuật toán đã dùng để mã hóa (HMAC SHA256 - HS256 hoặc RSA).

Ví dụ:

```
{
"alg": " SHA256",
"typ": "JWT"
}
```

Payload

Payload chứa các claims. Claims là một biểu thức về một thực thể (chẳng hạn user) và một số metadata phụ trợ. Có 3 loại claims thường gặp trong Payload: reserved, public và private claims.

Reserved claims: Đây là một số metadata được định nghĩa trước, trong đó một số metadata là bắt buộc, số còn lại nên tuân theo để JWT hợp lệ và đầy đủ thông tin: iss

(issuer), iat (issued-at time) exp (expiration time), sub (subject), aud (audience), và các thông tin khác.

Ví dụ:

```
{
  "iss": "scotch.io",
  "exp": 1300819380,
  "name": "Chris Sevilleja",
  "admin": true
}
```

Public Claims - Claims được cộng đồng công nhận và sử dụng rộng rãi.

Private Claims - Claims tự định nghĩa (không được trùng với Reserved Claims và Public Claims), được tạo ra để chia sẻ thông tin giữa 2 bên đã thỏa thuận và thống nhất trước đó.

Signature

Chữ ký (Signature) trong JWT là một chuỗi được mã hóa theo nguyên tắc sau:

```
HMACSHA256(
   base64UrlEncode(header) + "." +
   base64UrlEncode(payload),
   secret)
```

Do bản thân Signature đã bao gồm cả header và payload nên Signature có thể dùng để kiểm tra tính toàn vẹn của dữ liệu khi truyền tải.

Cấu trúc một JWT

Mỗi JWT là sự kết hợp bởi dấu (.) một đối tượng Header dưới định dạng JSON được mã hóa base64, một đối tượng payload dưới định dạng JSON được mã hóa base64 và một Signature cho URI cũng được mã hóa base64.

<base64-encoded header>.<base64-encoded payload>.<base64-encoded signature>

Ví dụ về một JWT:

“eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjEzODY4OTkxMzEsImlzcyI6ImppcmE6MTU0ODk1OTUiLCJxc2giOiI4MDYzZmY0Y2ExZTQxZGY3YmM5MGM4YWI2ZDBmNjIwN2Q0OTFjZjZkYWQ3YzY2ZWE3OTdiNDYxNGI3MTkyMmU5IiwiaWF0IjoxMzg2ODk4OTUxfQ.uKqU9dTB6gKwG6jQCuXYAiMNdfNRw98Hw_IWuA5MaMo”

2.5.2 Bảo mật với OAuth2

Giới thiệu Oauth:

OAuth là một phương thức chứng thực giúp các ứng dụng có thể chia sẻ tài nguyên với nhau mà không cần chia sẻ thông tin username và password. Từ Auth ở đây mang 2 nghĩa:

- Authentication: xác thực người dùng thông qua việc đăng nhập.
- Authorization: cấp quyền truy cập vào các Resource.

Lịch sử hình thành của Oauth:

- Năm 2006, Twitter đưa ra chuẩn OAuth đầu tiên có tên là OpenID, điểm yếu đó là yêu cầu người dùng phải cung cấp thông tin cá nhân (username + password).
- Năm 2010, phát hành phiên bản chính thức đầu tiên của Oauth 1.0 (RFC 5849).
- Sau đó lỗi bảo mật nghiêm trọng được phát hiện với tên gọi Session Fixation cho phép Hacker chiếm quyền truy cập vào tài nguyên của người dùng.
- Năm 2012, OAuth2 ra đời, tuy vẫn còn những lỗi bảo mật như dùng Chrome để Hack Facebook nhưng hiện vẫn đang được sử dụng khá rộng rãi.

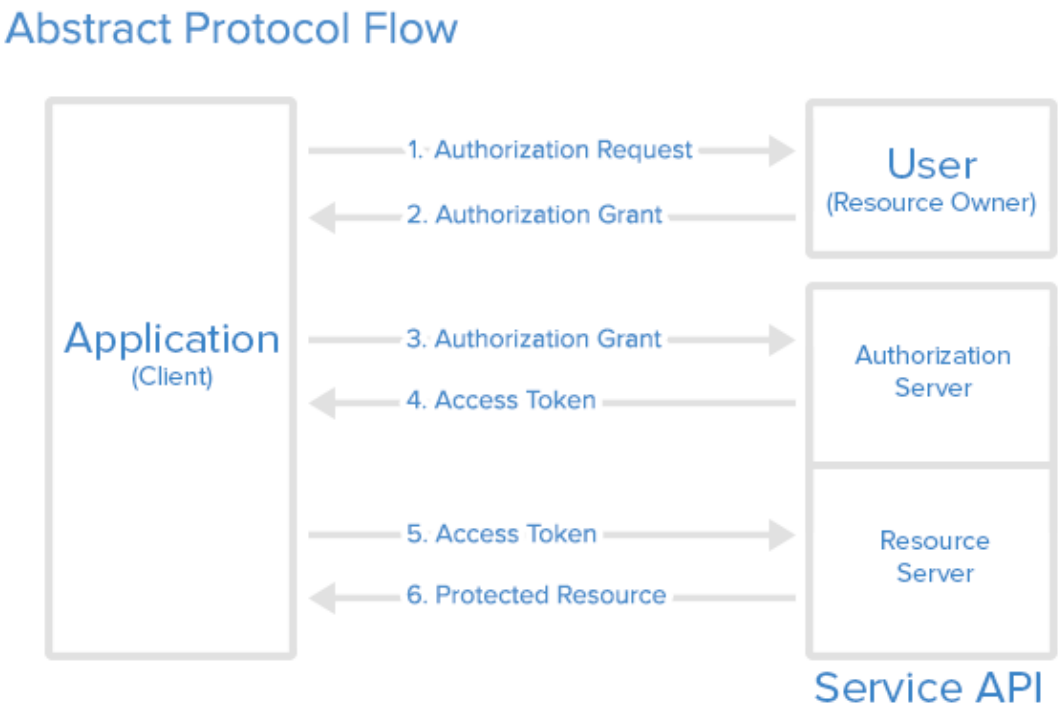
OAuth2 không đơn thuần chỉ là giao thức kết nối, nó là một "nền tảng" mà chúng ta phải triển khai ở cả hai phía: Client và Server. Sau đây hãy cùng làm quen với các tác nhân (hay đối tượng) tham gia vào hoạt động của OAuth2.

Các tác nhân (đối tượng) trong Oauth2:

OAuth2 làm việc với 4 đối tượng mang những vai trò riêng:

- Resource Owner (User): Là những người dùng ủy quyền cho ứng dụng cho phép truy cập tài khoản của họ. Sau đó ứng dụng được phép truy cập vào những dữ liệu người dùng nhưng bị giới hạn bởi những phạm vi (scope) được cấp phép. (VD: chỉ đọc hay được quyền ghi dữ liệu).
- Client (Application): Là những ứng dụng mong muốn truy cập vào dữ liệu người dùng. Trước khi được phép tương tác với dữ liệu thì ứng dụng này phải qua bước ủy quyền của User, và phải được kiểm tra xác nhận thông qua API.
- Resource Server (API): Nơi lưu trữ thông tin tài khoản của User và được bảo mật.
- Authorization Server (API): Làm nhiệm vụ kiểm tra thông tin user (VD: ID), sau đó cấp quyền truy cập cho Application thông qua việc phát sinh "access token".

Sơ đồ luồng hoạt động của Oauth2:



Hình 2.1. Sơ đồ luồng hoạt động của Oauth2

1. Application yêu cầu ủy quyền để truy cập vào Resource Server thông qua User.
2. Nếu User ủy quyền cho yêu cầu trên, Application sẽ nhận được giấy ủy quyền từ phía User (dưới dạng một token string nào đó chẳng hạn).
3. Application gửi thông tin định danh (ID) của mình kèm theo giấy ủy quyền của User tới Authorization Server.
4. Nếu thông tin định danh được xác thực và giấy ủy quyền hợp lệ, Authorization Server sẽ trả về cho Application access_token. Đến đây quá trình ủy quyền hoàn tất.
5. Để truy cập vào tài nguyên (resource) từ Resource Server và lấy thông tin, Application sẽ phải đưa ra access_token để xác thực.
6. Nếu access_token hợp lệ, Resource Server sẽ trả về dữ liệu của tài nguyên đã được yêu cầu cho Application.

Luồng hoạt động thực tế có thể sẽ khác nhau tùy thuộc vào việc ứng dụng sử dụng loại ủy quyền (authorization grant type) nào, trên đây chỉ là ý tưởng chung để thực hiện.

2.5.3 Lựa chọn giải pháp

Dịch vụ web kiến trúc REST càng phổ biến thì vấn đề bảo mật cũng trở nên quan trọng. Tuy nhiên tùy theo nhu cầu của hệ thống mà người thiết kế phải lựa chọn áp dụng giải pháp nào cho hợp lý. Với hai giải pháp được nêu trên đây gồm JSON Web Token và Oauth2 chúng ta có thể thấy:

JSON Web Token là một giao thức xác thực. Nó là một tập các quy định chặt chẽ các hướng dẫn trong việc ban hành và xác nhận của các thẻ truy cập. Các thẻ có yêu cầu được sử dụng bởi một ứng dụng để hạn chế quyền truy cập cho người dùng

Oauth2 là một khung (framework) xác thực bao gồm các hướng dẫn chi tiết, cho phép người sử dụng và các ứng dụng cho phép quyền cụ thể cho các ứng dụng khác trong cả hai môi trường cá nhân và công cộng.

Dựa trên những mô tả của hai phương thức này (2.5.1 và 2.5.2) tác giả lựa chọn JSON Web Token là phương pháp bảo mật cho API RESTful bởi các lợi thế sau:

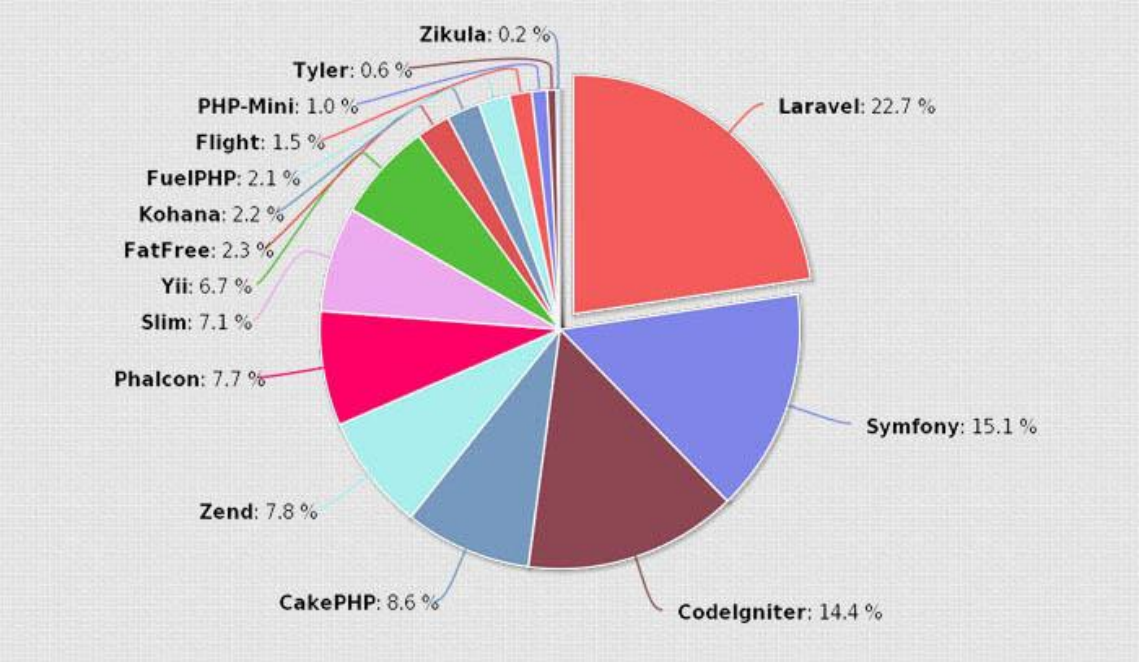
- Thời gian tìm hiểu nhanh
- Mã nguồn ngắn gọn
- Giảm bảo trì
- Giải pháp cho doanh nghiệp lớn.

CHƯƠNG 3: KHUNG LÀM VIỆC LARAVEL

3.1 Giới thiệu

Laravel là một khung làm việc PHP mã nguồn mở và miễn phí, được phát triển bởi Taylor Otwell và nhắm vào mục tiêu hỗ trợ phát triển các ứng dụng web theo kiến trúc model-view-controller (MVC). Những tính năng nổi bật của Laravel bao gồm cú pháp dễ hiểu – rõ ràng, một hệ thống đóng gói mô đun và quản lý gói phụ thuộc, nhiều cách khác nhau để truy cập vào các cơ sở dữ liệu quan hệ, nhiều tiện ích khác nhau hỗ trợ việc triển khai vào bảo trì ứng dụng.

Vào khoảng tháng 3 năm 2015, các lập trình viên đã có một cuộc bình chọn khung làm việc PHP phổ biến nhất, Laravel đã giành vị trí quán quân cho khung làm việc PHP phổ biến nhất năm 2015, theo sau lần lượt là Symfony, CodeIgniter, CakePHP, Zend vào một số khác. Trước đó, Tháng 8/2014, Laravel đã trở thành dự án PHP phổ biến nhất và được theo dõi nhiều nhất trên Github.



Hình 3.1. Tỷ lệ đánh giá các khung làm việc PHP

3.2 Lịch sử phát triển của Laravel

Laravel được Taylor Otwell tạo ra như một giải pháp thay thế cho CodeIgniter, cung cấp thêm nhiều tính năng quan trọng như xác thực và phân quyền. Laravel là một khung làm việc đơn giản, dễ hiểu, hỗ trợ lập trình viên hiện thực ý tưởng một cách nhanh nhất bằng nhiều tính năng hỗ trợ như Eloquent ORM mạnh mẽ, xác thực đơn giản, phân trang hiệu quả, và hơn thế nữa.

Bản Laravel beta đầu tiên được phát hành vào ngày 9/6/2011, tiếp đó là *Laravel 1* phát hành trong cùng tháng. Laravel 1 bao gồm các tính năng như xác thực, bản địa

hóa, model, view, session, định tuyến và các cơ cấu khác, nhưng vẫn còn thiếu controller, điều này làm nó chưa thật sự là một khung làm việc MVC đúng nghĩa.

Laravel 2 được phát hành vào tháng 9 năm 2011, mang đến nhiều cải tiến từ tác giả và cộng đồng. Tính năng đáng kể bao gồm hỗ trợ controller, điều này thực sự biến Laravel 2 thành một khung làm việc MVC hoàn chỉnh, hỗ trợ Inversion of Control (IoC), hệ thống khuôn mẫu Blade. Bên cạnh đó, có một nhược điểm là hỗ trợ cho các gói của nhà phát triển bên thứ 3 bị gỡ bỏ.

Laravel 3 được phát hành vào tháng 2 năm 2012, với một loạt các tính năng mới bao gồm giao diện dòng lệnh (CLI) tên "Artisan", hỗ trợ nhiều hơn cho hệ thống quản trị cơ sở dữ liệu, chức năng ánh xạ cơ sở dữ liệu Migration, hỗ trợ "bắt sự kiện" trong ứng dụng, và hệ thống quản lý gói gọi là "Bundles". Lượng người dùng và sự phổ biến tăng trưởng mạnh kể từ phiên bản Laravel 3.

Laravel 4, tên mã "Illuminate", được phát hành vào tháng 5 năm 2013. Lần này thực sự là sự lột xác của khung làm việc Laravel, di chuyển và tái cấu trúc các gói hỗ trợ vào một tập được phân phối thông qua Composer, một chương trình quản lý gói thư viện phụ thuộc độc lập của PHP. Bố trí mới như vậy giúp khả năng mở rộng của Laravel 4 tốt hơn nhiều so với các phiên bản trước. Ra mắt lịch phát hành chính thức mỗi sáu tháng một phiên bản nâng cấp nhỏ. các tính năng khác trong Laravel 4 bao gồm tạo và thêm dữ liệu mẫu (database seeding), hỗ trợ hàng đợi, các kiểu gửi mail, và hỗ trợ "xóa mềm" (soft-delete: record bị lọc khỏi các truy vấn từ Eloquent mà không thực sự xóa hẳn khỏi DB).

Laravel 5 được phát hành trong tháng 2 năm 2015, như một kết quả thay đổi đáng kể cho việc kết thúc vòng đời nâng cấp Laravel lên 4.3. Bên cạnh một loạt tính năng mới và các cải tiến như hiện tại, Laravel 5 cũng giới thiệu cấu trúc cây thư mục nội bộ cho phát triển ứng dụng mới. Những tính năng mới của Laravel 5 bao gồm hỗ trợ lập lịch định kỳ thực hiện nhiệm vụ thông qua một gói tên là "Scheduler", một lớp trừu tượng gọi là "Flysystem" cho phép điều khiển việc lưu trữ từ xa đơn giản như lưu trữ trên máy local – dễ thấy nhất là mặc định hỗ trợ dịch vụ Amazone S3, cải tiến quản lý assets thông qua "Elixir", cũng như đơn giản hóa quản lý xác thực với các dịch vụ bên ngoài bằng gói "Socialite".

3.3 Cấu trúc của Laravel

Laravel là một khung làm việc khá mới mẻ nhưng bù lại nó có tài liệu hướng dẫn sử dụng (trên trang chủ https://laravel.com/) đầy đủ, rõ ràng, dễ hiểu và nhiều ưu điểm hấp dẫn. Khung làm việc Laravel sử dụng cấu trúc MVC và trên nền tảng lập trình hướng đối tượng OOP đồng thời kế thừa được sức mạnh của các khung làm việc đi

trước và đem đến những tính năng mới của PHP 5.3 trở lên. Chúng ta hãy điểm qua một số tính năng mạnh mẽ của Laravel .

- Route trong Laravel thật sự khác biệt, mới mẻ và đầy mạnh mẽ.
- RESTFUL Controllers
- Eloquent ORM: đây là một ORM tuyệt vời với khả năng migration data và làm việc tốt với MySQL, Postgres, SQL Server và SQLite.

3.3.1 Route

Mục đích của router là định tuyến đến những controller cụ thể nào từ phía request của người sử dụng. Đây cũng là điều dễ thấy trong các khung làm việc PHP phổ biến hiện nay. Nhưng với khung làm việc Laravel, chúng ta có thể thấy được sự khác biệt rõ ràng với route. Laravel không cần thiết phải tạo ra controller mà thực thi những công việc mong muốn 1 cách trực tiếp tại route dễ dàng. Bởi kết hợp theo phương pháp đóng kín (closure) quen thuộc trong javascript để thực thi điều đó.

a. Routing bacsic

```
Route::get('/hello-world', function()
{
   return 'Hello World;
});
```

Một yêu cầu dạng HTTP GET được gửi đến địa chỉ http://laravel-app/hello-world và một giá trị trả về sẽ là một chuỗi "Hello World". Chúng ta, có thể tùy biến lên cao nhất với route một cách dễ dàng trong khung làm việc laravel. Đây là điều mà các khung làm việc PHP hiện nay vẫn chưa thể làm được một cách hoàn hảo như Laravel. Để làm việc được với route trước hết ta tiếp xúc chúng với công thức cơ bản như sau:

```
Route::method('Tên định danh',Tham số)
```

Method là get, post, any

- Post: dành cho các thao tác lấy từ form như thêm record.
- Get: dành cho các thao tác truy cập thông thường tương đương với request cơ bản trong PHP.
- Any: là sự tổng hợp của các thao tác ở trên.

Tham số: Những thao tác mà chúng ta mong muốn với định danh trên. Tham số có thể là hàm xử lý, có thể là array() chứa các thông tin xử lý khác, có thể là sự ám chỉ cụ thể một controller nào đó cho định danh.

```
Route::get("/home","HomeController@showWelcome");
```

Khi yêu cầu HTTP GET /home nó sẽ gọi HomeController với action là showWelcome

Route theo kiểu post, được sử dụng khi ta đẩy dữ liệu lên máy chủ, ví dụ: khi ta đăng ký user mới

```
Route::post('foo/bar', function()
{
        return 'Hello World';
});
```

Khi chúng ta muốn sử dụng cả hay phương thức trên thì chúng ta có thể sử dụng như sau:

```
Route::any('foo', function()
{
return 'Hello World';
});
```

Nếu bạn muốn gửi kèm một tham số, trong laravel ta định danh tham số dựa vào ký tự sau: {tên}. Và ở hàm thiết lập ta xem nó như đối số trong hàm.

```
//route có tham số
Route::get('user/{id}', function($id){
return 'User '.$id;
});
//route có tham số mặc định
Route::get('user/{name?}', function($name = 'John'){
return $name;
});
//Route ràng buộc bởi biểu thức.
Route::get('user/{name}', function($name){
//
})->where('name', '[A-Za-z]+');
Route::get('user/{id}', function($id){
//
})->where('id', '[0-9]+');
//truyền một mảng
Route::get('user/{id}/{name}', function($id, $name){
//
})->where(array('id' => '[0-9]+', 'name' => '[a-z]+'))
```

b. Route có lọc (filter)

Route có lọc là cái cách ta tạo ra khi muốn giới hạn quyền truy cập tới route nào, việc này rất hữu ích khi chúng ta phân quyền cho một người dùng nào đó, hay khi truy cập vào một trang nào đó thì cần phải chứng thực. Và lọc (filter) được khai báo tại app/filters.php.

```
Route::filter('old', function(){
   if (Input::get('age') < 200){
return Redirect::to('home');
   }
});
//Việc sử dụng filter cho route bằng cách như sau:
Route::get('user', array('before' => 'old', function(){
     return 'You are over 200 years old!';
}));
//áp dụng tới controller như sau
Route::get('user', array('before' => 'old', 'uses' =>
  'UserController@showProfile'));
//Có thể sử dụng nhiều filter trong một route
Route::get('user', array('before' => 'auth|old', function(){
     return 'You are authenticated and over 200 years old!';
}));
//Route theo tên:
Route::get('user/profile', array('as' => 'profile', function(){
//
}));
//Bạn cũng có thể chỉ tới action controller
Route::get('user/profile', array('as' => 'profile', 'uses' =>
  'UserController@showProfile'));
//Bạn cũng có thể sử dụng để gen ra một route hay route trực tiếp
$url = URL::route('profile');
$redirect = Redirect::route('profile');
//truy cập tới route đang chạy bằng phương thức
$name = Route::currentRouteName();
```

```
//Đôi khi bạn cần filters tới một nhóm của route, thay vì chỉ định tới từng filter của route thì ta sử dụng route theo nhóm
Route::group(array('before' => 'auth'), function(){
    Route::get('/', function(){
// Has Auth Filter
});
    Route::get('user/profile', function()
{
// Has Auth Filter
});
});
```

c. Sub-Domain Routing

```
Route::group(array('domain' => '{account}.myapp.com'), function(){
Route::get('user/{id}', function($account, $id){
//
});
});
```

d. Route theo tiền tố:

Một nhóm của route có thể được sử dụng trong một tiền tố nào đó

```
Route::group(array('prefix'=>'admin'),function(){
        Route::get('user',function(){
//
});
});
```

3.3.2 Controller

Một ví dụ về lớp controller cơ bản

```
class UserController extends Base Controller{
   /**
    * Show the profile for the given user.
    */
    public function showProfile($id){
    $user =User::find($id);
    return View::make('user.profile', array('user'=> $user));
}}
```

Tất cả những lớp controllers đều được mở rộng từ lớp BaseController. Lớp BaseController cũng được lưu trữ trong thư mục app/controllers. Bạn có thể route tới action controllers như sau trong file route. ta có thể gọi .

```
Route::get('user/{id}', 'UserController@showProfile');
```

RESTFUL Controllers

Laravel cho phép chúng ta sử dụng một route đơn mà xử lý mọi hành động trong controller

```
Route::controller('users', 'UserController');
```

Phương thức controller chấp nhận 2 đối số, đối số thứ nhất là đường dẫn URI, đối số thứ 2 là tên lớp của controller

```
class UserController extends BaseController{
public function getIndex(){
//
}
public function postProfile(){
//
}
}
```

Phương thức index sẽ được làm việc khi trên trình duyệt ta gõ users. Nếu chúng ta muốn sử dụng nhiều từ trong tên của controller action thì khi truy cập trình duyệt thêm dấu “-” vào. vd: users/admin-profile .

```
public function getAdminProfile() {}
```

Resource Controllers

Để lập một controller qua dòng lệnh, thì ta thực hiện

```
php artisan controller:make PhotoController
```

Bây giờ chúng ta có thể đăng ký một resourceful route tới controller

```
Route::resource('photo', 'PhotoController');
```

Việc lập route đơn này sẽ xử lý nhiều route của RESTful actions trên nguồn photo. Dưới đây là bảng ánh xạ giữa route và action

Verb	Path	Action	Route Name
GET	/resource	index	resource.index
GET	/resource/create	create	resource.create
POST	/resource	store	resource.store
GET	/resource/{resource}	show	resource.show
GET/resource/{resource}/edit	/resource/{resource}/edit	edit	resource.edit
PUT/PATCH	/resource/{resource}	update	resource.update
DELETE	/resource/{resource}	destroy	resource.destroy

Hình 3.2. Ánh xạ giữa route và action

Ngoài ra thì bạn có thể giới hạn được những action nào được thực thi

```
Route::resource('photo', 'PhotoController',array('only' => array('index', 'show')));
Route::resource('photo', 'PhotoController',array('except' => array('create', 'store', 'update')));
```

3.3.3 Eloquent ORM

Eloquent ORM là một trong những thế mạnh của khung làm việc Laravel mà một số khung làm việc khác không hỗ trợ được.

Tạo model User.php trong thư mục app/models/User.php

```
class User extends Eloquent {
    protected $table = 'my_users';
}
```

- Khi model được định nghĩa là chúng ta có thể thao tác trên nó,và lớp model đều phải kế thừa từ lớp Eloquent
- Thuộc tính $table sẽ khai báo bảng dữ liệu mà ta sẽ thao tác

a. Get dữ liệu

Truy vấn tất cả bản ghi

```
$users=User::all();
```

Truy vấn một bản ghi bởi khóa chính

```
$user = User::find(1);
var_dump($user->name);
```

Để đăng ký xử lý lỗi thì ta phải kích hoạt ModelNotFoundException

```
use Illuminate\Database\Eloquent\ModelNotFoundException;
App::error(function(ModelNotFoundException $e){
  return Response::make('Not Found', 404);
});
```

Truy vấn sử dụng Eloquent Models

```
$users = User::where('votes', '>', 100)->take(10)->get();
foreach ($users as $user){
var_dump($user->name);
}
```

b. Insert

Để tạo một bản ghi mới trong bảng CSDL, đơn giản chúng ta tạo một thực thể của model và gọi phương thức save

```
$user = new User();
$user->name = 'John';
$user->save()
```

Sử dụng phương thức Create

```
// Tạo một người dùng mới trong database
$user = User::create(array('name' => 'John'));
// Lấy người dùng bằng các thuộc tính, hoặc tạo người dùng nếu chưa tồn tại
$user = User::firstOrCreate(array('name' => 'John'));
// Lấy người dùng bằng các thuộc tính, hoặc tạo một bản ghi mới
$user = User::firstOrNew(array('name' => 'John'));
```

c. Update

Để cập nhật một Model, chúng ta có thể truy vấn nó, thay đổi thuộc tính và lưu nó lại:

```
$user = User::find(1);
$user->email = 'huytuan@framgia.com';
$user->save();
```

Đôi khi chúng ta muốn lưu không chỉ nó, mà toàn bộ những gì liên quan tới nó thì bạn sử dụng phương thức push

```
$user->push();
```

Chúng ta có thể chạy câu lệnh update để truy vấn model theo điều kiện

```
$affectedRows = User::where('id', '>', 100)->update(array('status' => 2));
```

d. Delete

Để xóa một bản ghi, đơn giản chúng ta gọi phương thức delete dựa vào khóa chính của dữ liệu :

```
$user = User::find(1);
$user->delete();
```

Xóa bằng khóa

```
User::destroy(1);
User::destroy(array(1, 2, 3));
User::destroy(1, 2, 3);
```

Ta cũng có thể xóa theo kiểu query

```
$affectedRows = User::where('id', '>', 100)->delete();
```

3.4 Bảo mật với Laravel

Trong thời đại internet bùng nổ, vấn đề bảo mật thật sự là vấn đề nhức nhối và bức bối. Hãy tìm hiểu khung làm việc Laravel 5 làm được những gì để bảo vệ trước những mối đe dọa đó. Các vấn đề sẽ được trình bày bao gồm:

- Cross-site Request Forgery (CSRF)
- Cross-site Scripting (XSS)
- SQL Injection

- Mass Assignment
- Cookies
- HTTPS

3.4.1 Giả mạo yêu cầu (Cross-site Request Forgery - CSRF)

CSRF là kiểu tấn công mượn cookie và session của bạn thực thi vài URL lên trang bị tấn công. Ví dụ: Bạn là admin của một web ABC và bạn đã đăng nhập vào. Bạn đi đọc báo, kiểm tra mail... và vẫn chưa đăng xuất trang web ABC. Bạn truy cập trang web kẻ tấn công, kẻ tấn công sẽ mượn quyền admin của bạn để thực thi ngầm lên trang web ABC của bạn.

Để phòng tránh kiểu tấn công này, Laravel kiểm tra token các yêu cầu không phải GET được gửi lên. Trong Laravel 5 đã tích hợp sẵn gói "HTML". Gói này hỗ trợ, khi bạn sử dụng *Form::open* hay *Form::model* đều tự động thêm vào 1 trường input ẩn tên là _token, input _token này được gửi cùng với dữ liệu form và được kiểm tra qua middleware. Nếu không sử dụng gói này, bạn có thể sử dụng hàm csrf_token để lấy token này để tự thao tác. Lưu ý, bạn cũng tránh dùng phương thức GET làm thay đổi dữ liệu CSDL.

3.4.2 Kịch bản lệnh (Cross-site Scripting (XSS)

XSS không tấn công vào CSDL hệ thống mà trực tiếp tấn công từ phía người dùng. Lỗi này xuất hiện khi ứng dụng cho phép kẻ tấn công đặt đoạn mã Javascript.

Ví dụ: Bạn cho phép người dùng đăng bình luận, bình luận sẽ được chèn vào trong CSDL. Sau đó bình luận được lấy ra hiển thị cho các người dùng khác. Thử nghĩ nếu bình luận đó là một đoạn Javascript và có thể thực thi, nó có thể lấy cookie và session của người dùng gửi về cho kẻ tấn công.

Để tránh được kiểu tấn công này, ta trên cương vị một lập trình viên không tin bất cứ dữ liệu nào mà người dùng gửi lên. Và trong các trang view (Blade template) hãy sử dụng {{ $var }} thay vì {!! $var !!} với những biến nào bạn nghi ngờ có thể xảy ra XSS.

{{ $var }} tương tự như echo htmlentities($var).

3.4.3 Nhúng câu lệnh SQL (SQL Injection)

Lỗi SQL Injection rất nặng có thể gây biến đổi CSDL hay mất cả quyền kiểm soát CSDL. Lỗi này xuất hiện khi thao tác với CSDL với những biến sơ ý chưa được lọc kỹ càng.

Mặc định, Laravel sẽ bảo vệ chúng ta khỏi tấn công kiểu này khi sử dụng hai thư viện Fluent Query builder và Eloquent ORM đều sử dụng PHP Data Object (PDO).

PDO sử dụng lệnh chuẩn bị trước cho phép chúng ta sử dụng các tham số chưa được thêm mã thoát.

Trong một vài trường hợp chúng ta muốn sử dụng một câu SQL phức tạp, chúng ta phải hết sức cẩn thận. Hãy xem đoạn code này:

```
DB::select(DB::raw("SELECT * FROM users WHERE name = $name"));
```

Quá nguy hiểm, biến $name có thể chưa được kiểm soát chặt chẽ và có khả năng gây ra lỗi, chúng ta có thể viết lại bằng cách truyền tham số để tránh SQL Injection:

```
DB::select(DB::raw("SELECT * FROM users WHERE name = ? ", [$name]));
```

3.4.4 Phép gán ồ ạt (Mass Assignment)

Mass Assignment là một lỗ hổng bảo mật, xuất hiện khi chúng ta dùng một mảng để tạo ra đối tượng của một Model:

```
$user = new User(Input::all());
```

Laravel sẽ cảnh báo cho bạn biết lỗi Mass Assignment. Để hiểu rõ hơn về lỗi này chúng ta tìm hiểu ví dụ sau:

Ta có model User được tạo như sau:

```
Schema::create('User', function($table){
  $table->increments('id');
  $table->string('name');
  $table->string('email');
  $table->boolean('is_admin');
});
```

Trong model User trên có trường is_admin để xác định quyền quản trị. Ta cho người dùng đăng ký tài khoản và đưa vào trường is_admin = 1, họ trở thành quản trị. Laravel đã cảnh báo cho chúng ta hãy thêm thuộc tính $fillable hoặc $guarded vào trong Model của chúng ta.

```
class User extends Eloquent{
  public $timestamps = false;
  protected $guarded = ['role'];
}
```

3.4.5 Cookies

Laravel hỗ trợ thao tác dễ dàng với cookies và tất cả các cookies chúng ta sử dụng đã được nhận dạng và mã hóa. Điều này có nghĩa là:

- Nếu cookie bị sao chép thì Laravel tự động loại bỏ chúng
- Chúng ta không thể sử dụng Javascript ở phía khách để đọc chúng

3.4.6 HTTPS

Nếu website của chúng ta sử dụng phương thức HTTP, chúng ta cần nhớ trong đầu là từng bit dữ liệu được trao đổi (cả mật khẩu) đều gửi ở dạng bản rõ (cleartext). Kẻ tấn công cùng chung mạng (network) có thể chặn và lấy các thông tin bí mật của phiên làm việc rồi đăng nhập như nạn nhân. Cách duy nhất để chống lại điều này là sử dụng HTTPS. Nếu chúng ta đã cài chứng chỉ SSL trên máy chủ web, Laravel có một vài hỗ trợ cho việc chuyển từ http:// sang https:// ở các routes xác định, chúng ta có thể tạo một middleware Secure để làm việc này với đoạn code dưới đây:

```
class Secure implements Middleware{
    public function handle($request, Closure $next) {
      if (!$request->secure() && app()->environment('production')) {
            return redirect()->secure($request->getRequestUri());
      }
      return $next($request);
    }
}
```

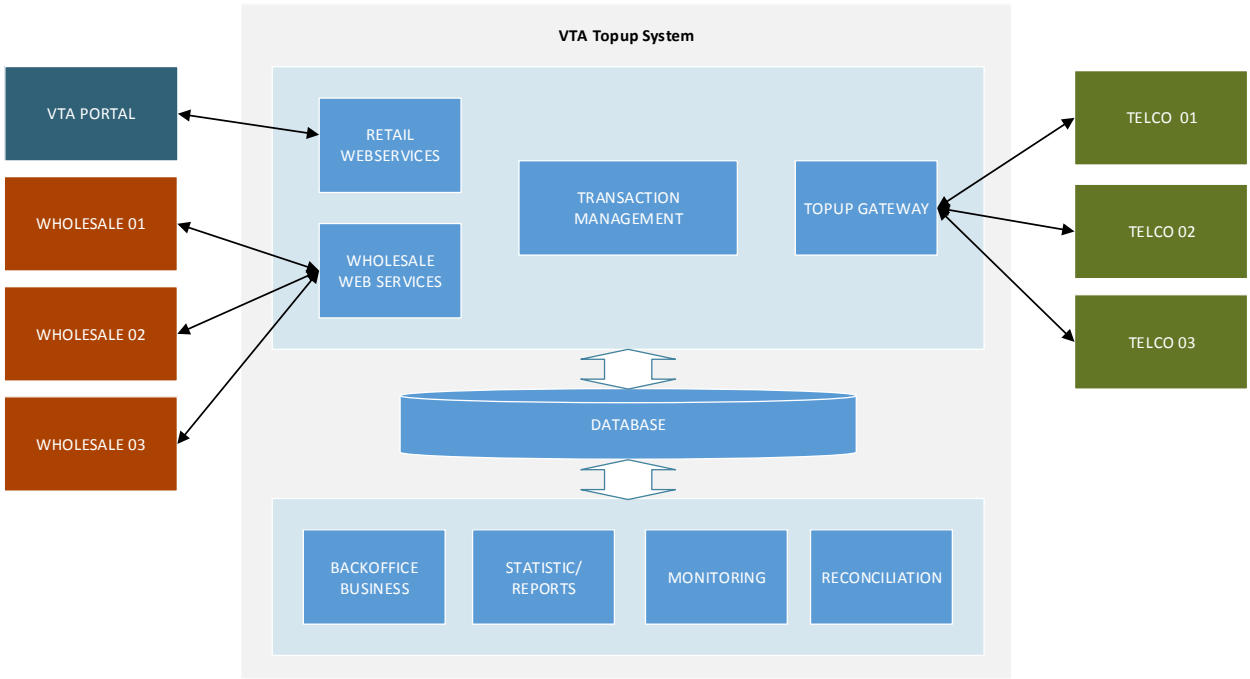
CHƯƠNG 4: THIẾT KẾ VÀ THỰC HIỆN HỆ THỐNG API TOPUP

Trong chương này sẽ giới thiệu tổng quan về hệ thống cũng như lược đồ, cấu trúc của hệ thống TOPUP và các công nghệ quan trọng được sử dụng trong hệ thống. Cũng trong chương này sẽ mô tả quá trình thiết kế REST API, mô tả cấu trúc REST API sẽ xây dựng, mô tả mối liên hệ giữa các mô đun, hơn nữa sẽ giải thích nguyên tắc REST áp dụng vào hệ thống như thế nào, ngoài ra còn mô tả lược đồ use case chính được sử dụng.

4.1 Giới thiệu hệ thống TOPUP

Hệ thống TOPUP [1] cung cấp cho khách hàng phương án nạp tiền trực tiếp vào tài khoản thuê bao trả trước, trả sau, tài khoản game, học trực tuyến,… bằng các thao tác đơn giản trên điện thoại, máy tính hoặc các thiết bị khác có kết nối internet, GPRS, Wifi hoặc 3G. Bạn có thể nạp phí cho tất cả các loại dịch vụ từ các nhà cung cấp dịch vụ. Ở trong nội dung luận văn này tác giả trình bày tính năng nạp tiền điện thoại thông qua hệ thống **VTA TOPUP.** Lợi ích khi sử dụng hệ thống **VTA TOPUP**:

- Nhanh chóng và tiện lợi: Chỉ cần thao tác đơn giản trên máy di động
- Liên tục và tức thì: Dịch vụ phục vụ tự động mọi lúc 24/7, bất cứ nơi đâu.
- Chia sẻ và quản lý: Không chỉ nạp tiền cho mình, khách hàng còn có thể nạp tiền cho người thân.



Hình 4.1 Sơ đồ tổng quan hệ thống VTA Topup

Trong đó:

- VTA Portal là web portal gọi truy vấn API thực hiện dịch vụ Topup

- Retail WS là Gateway đáp ứng kết nối các kênh thanh toán / topup trực tiếp của người dùng.
- Wholesale WS là Gateway đáp ứng kết nối các kênh thanh toán / topup của các đối tác bán buôn cước viễn thông.
- Transaction Management là mô đun quản lý giao dịch.
- Topup Gateway là cổng kết nối dịch vụ topup với các nhà mạng (NATCOM, MOVITEL,..).

4.2 Nguyên tắc hoạt động

Để thực hiện giao dịch Topup, khách hàng sẽ truy cập vào website có hệ thống Topup. Website này được cung cấp một bộ API Topup, gồm các công cụ để truy vấn thông tin: lấy thông tin tài khoản, số dư hiện tại, lịch sử giao dịch… và thực hiện giao dịch Topup. Khi thực hiện các giao dịch trên website sẽ gọi tương ứng các API cho mỗi giao dịch, hệ thống TopUp sẽ tiếp nhận các yêu cầu đó, sau đó phân tích các yêu cầu, xử lý các yêu cầu sau đó trả lại cho khách hàng một kết quả HTTP, kết quả đó có thể là kết quả yêu cầu của khách hàng, cũng có thể là một thông báo của hệ thống.

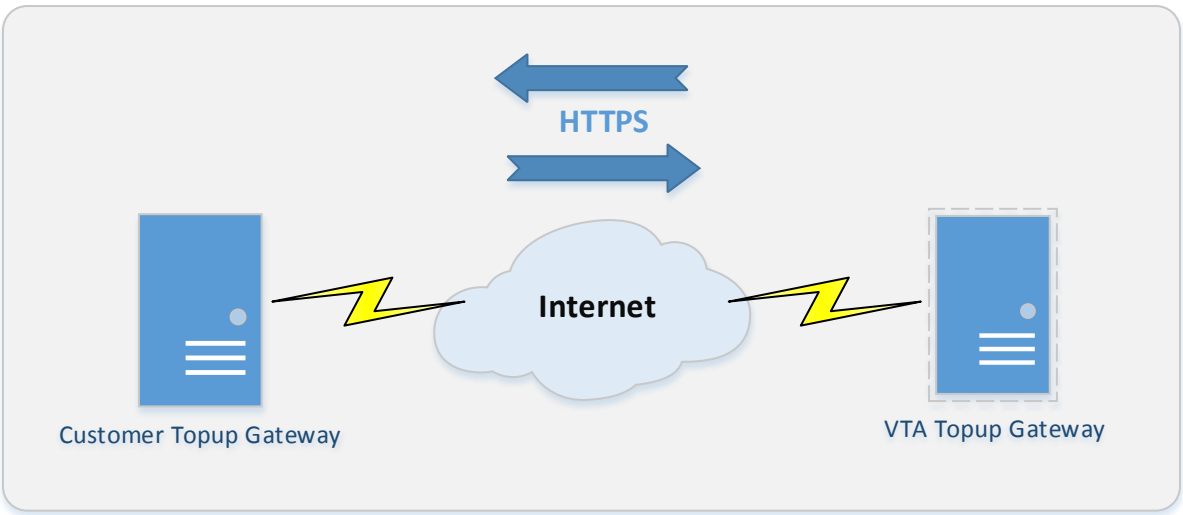
4.3 Tổng quan về hệ thống VTA TOPUP API

4.3.1 Tổng quan về APIs

Mô tả	Chi tiết
Giao thức	Vận chuyển bằng HTTP và sử dụng định dạng JSON để mô tả dữ liệu
Bảo mật và mã hóa	Tất cả các giao dịch được bảo vệ dành riêng cho các đối tác ủy quyền - Mã hóa dữ liệu thông qua SSL - Yêu cầu lọc địa chỉ IP - Xác thực tin nhắn HMAC SHA256 Client-ID, Request Timestamp và API Token
URI thử nghiệm	POST http://vta-address:vta-port/topup
Tài khoản thử nghiệm	- Client-ID: tester - Token: EXAMPLETOKENKEY

4.3.2 Kết nối

Dịch vụ VTA Top Up hỗ trợ kết nối qua Web API công cộng và hoạt động thông qua giao thức HTTP (HTTPS) và một URI cụ thể.



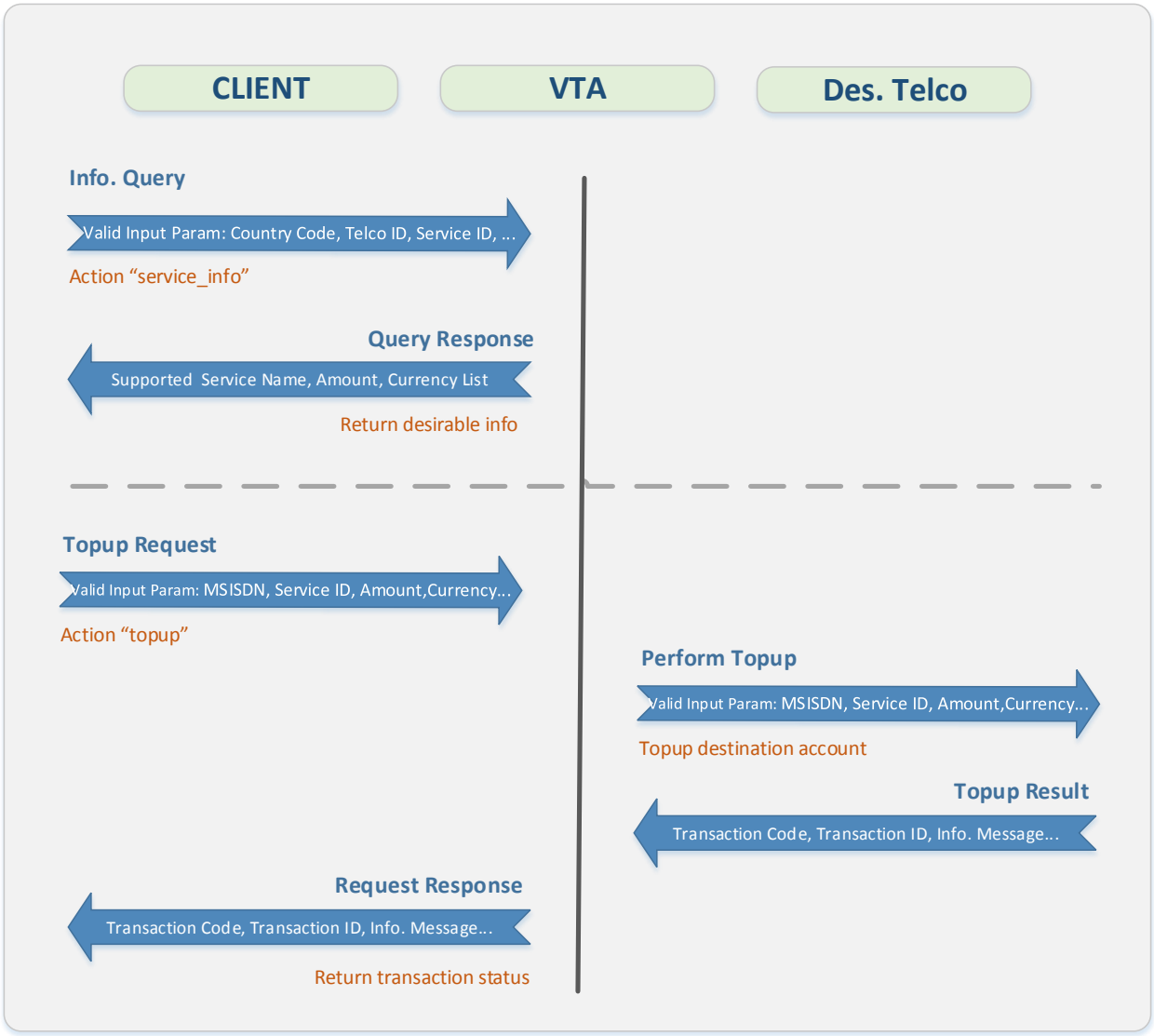
Hình 4.2 Kết nối của dịch vụ VTA Topup

Đối với một giao tiếp dữ liệu an toàn, VTA khuyến cáo khách hàng thiết lập một kết nối VPN IPSEC tới VTA Topup Gateway

4.3.3 Luồng hoạt động của TOPUP

Hệ thống này về cơ bản hỗ trợ 2 luồng quá trình:

- Thông tin truy vấn: để lấy thông tin tài khoản, số dư hiện tại, lịch sử giao dịch cũng như các dịch vụ topup có sẵn và số tiền topup mà bạn có thể sử dụng (ví dụ: trả trước điện thoại di động, ADSL, D-COM tài khoản 3G ...)
- Thực hiện Topup: thực hiện yêu cầu topup để nạp tiền cho một tài khoản đích.
- Quá trình dòng được mô tả như trong sơ đồ dưới đây:



Hình 4.3 Luồng hoạt động của hệ thống VTA Topup

4.3.4 Giao thức TCP/IP

Giao dịch topup được thực hiện trong thời gian thực bằng một kết nối TCP với một yêu cầu (request) và một thông điệp HTTP trả về (response). Thông điệp trả về chứa dữ liệu chứng thực VTA là bên gửi và hỗ trợ kiểm tra tranh chấp phí sau này. Hầu hết các giao dịch VTA Topup được xử lý trong khoảng 20 giây, nhưng có thể bị trì hoãn trong website của nhà mạng. Vì lý do đó VTA khuyến cáo khách hàng thiết lập thời gian chờ lớn nhất 300 giây. Mỗi giao dịch hơn 300 giây thì bị chấm dứt và không làm thay đổi số dư của tài khoản khách hàng.

4.3.5 Giao thức HTTP

Hệ thống VTA TOPUP sử dụng HTTP Header để thực hiện xác thực và phương thức POST để gửi yêu cầu dưới định dạng JSON

Thông điệp yêu cầu được định dạng như dưới đây:

Header	Content-Type: application/json Client-ID: <alphanumeric string> Timestamp: <interger> Signature: <string>

Payload	{
	“param_1” : “value_1”,
	…
	“param_n” : “value_n”
	}

Thông điệp trả về được định dạng

Header	VTA-Timestamp: <interger> VTA-Signature: <string>

Payload	{
	“code” : “value_code”,
	“message” : “value_message”,
	“param_1” : “value_1”,
	…
	“param_n” : “value_n”
	}

4.3.6 Bảo mật và xác thực

Các giao dịch của khách hàng được xác thực dựa trên mã khách hàng (Client ID) và HTTP Header của thông điệp yêu cầu. Mỗi thông điệp yêu cầu phải cung cấp các trường sau đây.

HTTP Request Header	Mô tả
Client-ID: <alphanumeric string>	ID của khách hàng
Timestamp: <interger>	Mốc thời gian (Timestamp) yêu cầu tính bằng mili giây (miliseconds)
Signature: <string>	Chuỗi Hash là sự kết hợp của Client-ID, Timestamp, Payload của yêu cầu và Client Token

Và thông điệp trả về cung cấp các trường sau đây:

HTTP Response Header	Mô tả
VTA-Timestamp: <interger>	Mốc thời gian (timestamp) của trả lời

	tính bằng mili giây (miliseconds)
VTA-Signature: <string>	Chuỗi Hash, là sự kết hợp Request Header Client-ID và Timestamp, Payload của Response, Timestamp của Response và Client Token

- "Client-ID": ID của khách hàng được cung cấp bởi VTA.
- "Timestamp": Mốc thời gian trong định dạng mili giây (milisecond), không có dấu thập phân. Ví dụ "1388983148.838" "." -> "1388983148838". Mốc thời gian của mỗi giao dịch là giá trị duy nhất tương ứng với mỗi Client-ID và một địa chỉ IP.
- "Signature": Mã hóa Base64 chuỗi băm SHA256, xác thực nội dung yêu cầu, được tạo bởi HMAC SHA256với khóa là Client Token được cung cấp bởi VTA .

```
signature = base64_encode
          (
              hash_hmac
              (
                Sha256,
                client-id.timestamp.payload,
                token_key
              )
            )
```

- "VTA-Signature": Được tạo ra giống với "Signature" với nội dung của thông điệp trả về và thông điệp yêu cầu thay thế

```
vta-signature = base64_encode
                (
                  hash_hmac
                  (
                    Sha256,
                    client-id.timestamp.payload.vta-timestamp,
                    token_key
                  )
                )
```

Ví dụ:

- Thông tin khách hàng được cung cấp bởi VTA

```
Client-ID: client
Token-key: CZB2IUkXIFpHJiQJ
```

- Timestamp tại thời điểm thực hiện request

```
Timestamp: 1389120142709
```

- Nội dung của thông điệp request

```
{"action":"ping"}
```

- Signature được tạo ra bằng công thức được mô tả bên trên

```
Signature = ODU1MGE1MGI4ZjAzN2Y3ODQzMDYzNDE5NTc5M2UzODJmYTI0Mjc5OQ==
```

- Nội dung thông điệp response

```
{"code":"0", "message":"Successful Request"}
```

- VTA-Signature tương ứng với VTA-Timestamp

```
VTA-Timestamp = 1389121066521,
VTA-Signature = YTU5ZDBjY2FlNzIyZmM5MTZkN2NmNzczYTk5MzJiNjFlNWZmYjllOQ==
```

4.4 Áp dụng kiến trúc REST

Thống kê tất cả các ưu điểm và nhược điểm ở trên thì một kiểu kiến trúc REST đã được đề xuất để xem xét nó có phải là một giải pháp thích hợp hay không, để thay thế API sử dụng XML-RPC bằng một API REST chung chung xử lý tất cả các dịch vụ, và cuộc điều tra về kiến trúc REST đã đưa ra được nhận định sau:

- Hiểu nguyên tắc của REST, thuộc tính gì, lợi ích và không thuận lợi gì và nó có thể được áp dụng để giải quyết bài toán đã nêu ở trên hay không.
- Hiểu được kiểu dữ liệu mà hệ thống VTA TOPUP đang xử lý, tài nguyên nào là quan trọng nhất và chúng chứa những thông tin gì.
- Đặt ra một phương pháp để cấu trúc sự truy cập dữ liệu sử dụng REST API và thiết kế của nó sẽ được xây dựng như thế nào, mô đun nào sẽ được xây dựng, với mỗi mô đun chúng sẽ là gì và chúng tương tác với các mô đun khác như thế nào.
- Thực thi REST API theo kết quả đã thiết kế theo nguyên mẫu ban đầu và sau đó tích hợp với nguyên mẫu trong hệ thống VTA TOPUP.
- Kiểm thử API bằng cách truy cập thông qua các từ khóa chính của GUI và thông qua dòng lệnh để kiểm tra cả hai API đều có thể sử dụng được.

4.4.1 Tài nguyên

Hoạt động đầu tiên của quá trình thiết kế cũng là một hoạt động khá quan trọng đó là xác định tài nguyên mà sẽ được truy cập thông qua API, điều này có thể làm được bằng cách xem xét các hoạt động của hệ thống VTA Topup Gateway, trong hệ thống này có hai tài nguyên chính đó là thông điệp Request là nội dung yêu cầu giao dịch và thông điệp Response là nội dung trả về của hệ thống.

Chúng ta sẽ mô tả chi tiết hơn về hai loại tài nguyên này và cũng mô tả xem chúng chứa những dữ liệu gì, cần có những tham số gì để tạo ra các tài nguyên này trong hệ thống dùng để thực hiện giao dịch trên hệ thống VTA Topup Gateway.

Như mô tả trong phần 4.3.5 hệ thống VTA Topup sử dụng HTTP Header để thực hiện xác thực và HTTP Payload có định dạng JSON để chứa thông tin giao dịch.

Thông điệp Request được định dạng như dưới đây:

Header	Content-Type: application/json Client-ID: <alphanumeric string> Timestamp: <interger> Signature: <string>

Payload	{
	“param_1” : “value_1”,
	…
	“param_n” : “value_n”
	}

Thông điệp Response được định dạng:

Header	VTA-Timestamp: <interger> VTA-Signature: <string>

Payload	{
	“code” : “value_code”,
	“message” : “value_message”,
	“param_1” : “value_1”,
	…
	“param_n” : “value_n”
	}

Ở thông điệp Request HTTP Header chứa: Client-ID, Timestamp, Signature. Ở thông điệp Response HTTP Header chứa: VTA-Timestamp, VTA-Signature được mô tả ở phần 4.3.6

Đối với phần HTTP Playload do VTA Topup Gateway cung cấp nhiều API, mỗi API có một chức năng riêng nên thông số HTTP Playload sẽ có những sự khác biệt. Tuy nhiên, cấu trúc chính của chúng bao gồm:

Đối với thông điệp Request HTTP Playload chứa các trường sau:

"param_1", "param_2", "param_3"...: Đây là các trường giá trị tương ứng với mỗi hành động cần các thông tin phụ để mô tả yêu cầu chi tiết

Đối với thông điệp Response HTTP Playload chứa các trường sau:

- "code": Đây là thông tin trả về của hệ thống về tình trạng của giao dịch.
- "message": Mô tả chi tiết về tình trạng giao dịch.
- "param_1", "param_2", "param_3" .. : Đây là các thông tin chi tiết về yêu cầu của mỗi giao dịch mà hệ thống trả về.

4.4.2 Đánh địa chỉ

Một trong số nguyên tắc chính của REST đó là mỗi tài nguyên được đánh địa chỉ bởi một URI duy nhất như đã thảo luận ở phần 1.5.2. Để thực hiện yêu cầu này thì trong REST API sẽ thiết kế URI duy nhất và dễ dàng phân biệt với mỗi loại tài nguyên.

- Phần thứ nhất của URI chỉ ra đây là hành động nào, như vậy phần thứ nhất có thể phân biệt rõ các API.

Ví dụ:

http://vta-address:vta-port/ping : Kiểm tra tình trạng của API

http://vta-address:vta-port/check-wallet : Kiểm tra số dư tài khoản

http://vta-address:vta-port/service-info: Trả về thông tin về các dịch vụ topup

- Nếu URI có phần thứ hai, thì phần thứ hai xác định tài nguyên thuộc loại mà tài nguyên phần thứ nhất của URI đã xác định.

Ví dụ:

http://vta-address:vta-port/ service-info/countries: trả về một danh sách của tất cả các nước có sẵn mà khách hàng đã đăng ký

Với ví dụ cơ bản như trên thì ta có thể hiểu và dễ dàng sử dụng cách đánh địa chỉ trực tiếp xác định một tài nguyên.

4.4.3 Phi trạng thái

Như đã mô tả trong phần 1.5.3, thì mọi yêu cầu HTTP thì đều xảy ra trong một tình huống hoàn toàn riêng biệt và không có thông tin về máy khách lưu trữ trên máy chủ từ yêu cầu này đến yêu cầu khác. Với hệ thống VTA Topup không lưu trữ trạng thái của bất kỳ giao dịch nào. Tất cả các giao dịch đều phải thực hiện xác thực bằng cách truyền thông các thông tin Client-ID, Timestamp, Signature ở phần HTTP

Header. Điều này giúp hệ thống dễ phát triển,bảo trì, mở rộng vì không cần tốn công lưu trữ trạng thái của phía khách. Hệ thống phát triển theo hướng này có ưu điểm nhưng cũng có khuyết điểm là gia tăng lượng thông tin cần truyền tải giữa khách và máy chủ.

4.4.4 Liên kết với nhau

Ứng dụng liên kết cho phép người dùng liên kết đến các tài nguyên khác thông qua API. Tài nguyên phải được kết nối với tài nguyên khác và trong đại diện của tài nguyên có gắn các liên kết đến các tài nguyên khác. Đây là một trong những tính năng cơ bản mà sẽ được cung cấp trong API được cài đặt trong luận văn này, theo đó với sự liên kết này thì:

- Mỗi danh sách tài nguyên Request, Response đều có chứa các liên kết đến các tài nguyên liên quan nếu có.
- Trong tài nguyên Request thì có chứa liên kết đến tài nguyên Response, là những Response thuộc Request này.
- Trong tài nguyên Response thì có chứa liên kết đến tài nguyên Request, là Request chứa Response này.

4.4.5 Giao diện đồng nhất

Hầu hết các API được cung cấp trong các hệ thống dựa trên nền web ngày nay thì chỉ sử dụng hai phương thức GET và POST để thực thi các thao tác khác nhau trên các tài nguyên của hệ thống, các thao tác đó là receive, create, update hoặc delete. Tương ứng với các thao tác này thì trong REST API sử dụng các hành động tương ứng với các thao tác trên là GET, POST, PUT và DELETE như đã mô tả trong phần 1.5.5. Bởi vậy trong REST API bốn phương thức đó sẽ được sử dụng để thực hiện các chức năng mà ứng dụng dự định thiết kế.

Chúng ta sẽ mô tả chi tiết hơn chức năng của bốn phương thức này với các yêu cầu trên HTTP:

- **/resource**
  - + GET: hiển thị danh sách tài nguyên trên hệ thống, nếu URI có yêu cầu các tham số truy vấn thì danh sách tài nguyên trả về sẽ phù hợp với khóa được chỉ định trong URI. Nếu kết quả trả về là dạng XHTML thì trong kết quả đó có chứa liên kết đến các tài nguyên khác.
  - + POST: thêm mới một tài nguyên, dữ liệu được truyền vào dạng form
  - + PUT: không ứng dụng.
  - + DELETE: không ứng dụng.
- **/resource/X (X là định danh xác định tài nguyên)**

+ GET: trả về thông tin của tài nguyên với định danh X.
+ POST: không ứng dụng.
+ PUT: không ứng dụng.
+ DLETE: không ứng dụng.

- **/resource/search**

+ GET: trả về một form cho phép tìm kiếm tài nguyên, cho phép điền các trường thông tin trên form phù hợp với các trường thông tin mà người dùng muốn tìm kiếm, khi submit form một yêu cầu GET sẽ được gửi tới một URI mà bao gồm các tiêu chí tìm kiếm, chẳng hạn như /search?name=sonnt.
+ POST: không ứng dụng.
+ PUT: không ứng dụng.
+ DLETE: không ứng dụng.

- **/resource/new**

+ GET: trả về một form cho phép tạo mới tài nguyên, form này gồm các trường thông tin phụ thuộc vào tài nguyên mà người dùng muốn tạo, khi submit form thì một yêu cầu POST được gửi tới /resource với giá trị các trường thông tin được đính kèm theo.
+ POST: không ứng dụng.
+ PUT: không ứng dụng.
+ DLETE: không ứng dụng.

4.4.6 Khả năng cache

Cache là một trong số nguyên tắc quan trọng của REST. Mục đính là cache được tài nguyên nhiều nhất có thể để giảm tải cho máy chủ, thông thường nếu tài nguyên là các dạng danh sách thì không cache, vì danh sách thường hay thay đổi, tuy nhiên cũng có một số trường hợp tập tài nguyên là không thay đổi thì vẫn có thể được cache.

Nói cách khác, các tài nguyên có dạng như /service_info sẽ được cache. Để cache tài nguyên này thì cơ chế validation của HTTP sẽ được sử dụng. Mỗi lần máy khách nhận thông tin với mã service_info sử dụng phương thức GET, thì máy chủ sẽ kiểm tra nếu tài nguyên đã được thay đổi kể từ khi phiên bản cuối mà máy khách yêu cầu thì máy chủ sẽ làm một nhiệm vụ là lưu lại thời gian thay đổi lần cuối với mỗi tài nguyên riêng trong hệ thống. Nếu tài nguyên không thay đổi thì hệ thống sẽ trả về mã 304 (Not Modified) của HTTP với nội dung rỗng, điều này có nghĩa là lượng công việc mà bên phía máy chủ phải làm là cực tiểu, nói cách khác nếu tài nguyên được thay đổi thì máy chủ sẽ trả về một kết quả và ghi đè thời gian thay đổi lên thời gian hiện tại.

4.5 Thiết kế chi tiết các API

Hiện tại hệ thống VTA Topup hỗ trợ 5 API, mỗi API có được xác định bởi một URI riêng biệt. Mỗi API, tùy thuộc vào từng yêu cầu cụ thể mà bắt buộc các thông tin kèm theo khi thực hiện giao dịch hoặc hệ thống sẽ từ chối yêu cầu.

URI	Mô tả
GET /ping	Kiểm tra tình trạng của API
GET /check-wallet	Lấy thông tin tài khoản và số dư hiện tại
GET /service-info	Lấy thông tin dịch vụ, giá, phí của công ty viễn thông và các quốc gia tương ứng
POST /topup	Thực hiện hành động Topup
GET /trans-history	Lấy lịch sử giao dịch

4.5.1 Phương thức "Ping"

API này được sử dụng để kiểm tra tình trạng kết nối của khách hàng và hệ thống VTA. Hình 4.3 chỉ luồng khi mà người dùng cần kiểm tra kết nối của hệ thống, máy khách sẽ gửi một yêu cầu HTTP GET tới URI /ping. Khi đó yêu cầu sẽ được hệ thống VTA tiếp nhận. Đầu tiên mô đun xác thực sẽ phân tích URI và các tham số header để xác thực người dùng như đã nêu trong phần 4.3.6. Trong trường hợp người dùng được xác thực hệ thống sẽ thực hiện kiểm tra kết nối đến nhà mạng cung cấp dịch vụ theo cú pháp mà nhà mạng cung cấp. Kết quả nhà mạng gửi về sẽ được tạo ra một kết quả HTTP trả lại cho người dùng.

Định dạng request

URI: GET http://vta-address:vta-port/ping

Định dạng response

Trường	Kiểu dữ liệu	Mô tả
code	interger	Mã lỗi của request
message	string	Chú thích cho mã lỗi

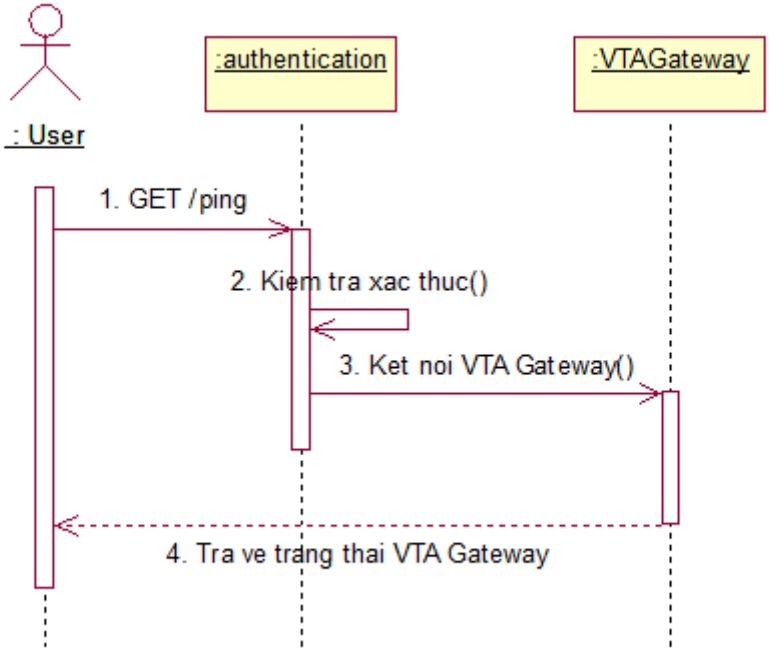
Ví dụ:

Request

Uri: http://vta-address:vta-port/ping

Response

```
{
"code":"0",
"message":"Successful Request"
}
```



Hình 4.4. Lược đồ tuần tự API Ping

4.5.2 Phương thức "Check Wallet"

Phương thức này dùng để khách hàng truy vấn số dư hiện tại. Hình 4.4 chỉ ra luồng khi mà người dùng muốn truy vấn các thông tin của tài khoản như: số dư, giới hạn thành toán, đơn vị tiền tệ... Để thực hiện hành động này máy khách sẽ gửi một yêu cầu HTTP GET tới URI /check-wallet. Khi đó yêu cầu sẽ được hệ thống VTA tiếp nhận. Đầu tiên môđun xác thực sẽ phân tích URI và các tham số header để xác thực người dùng như đã nêu trong phần 4.3.6. Trong trường hợp người dùng được xác thực hệ thống sẽ yêu cầu thông tin người dùng trong CSDL và tạo ra một kết quả HTTP gửi lại cho người dùng.

Định dạng request

URI: GET http://vta-address:vta-port/check-wallet

Định dạng response

Trường	Kiểu dữ liệu	Mô tả
code	Interger	Mã lỗi của request
message	String	Chú thích cho mã lỗi

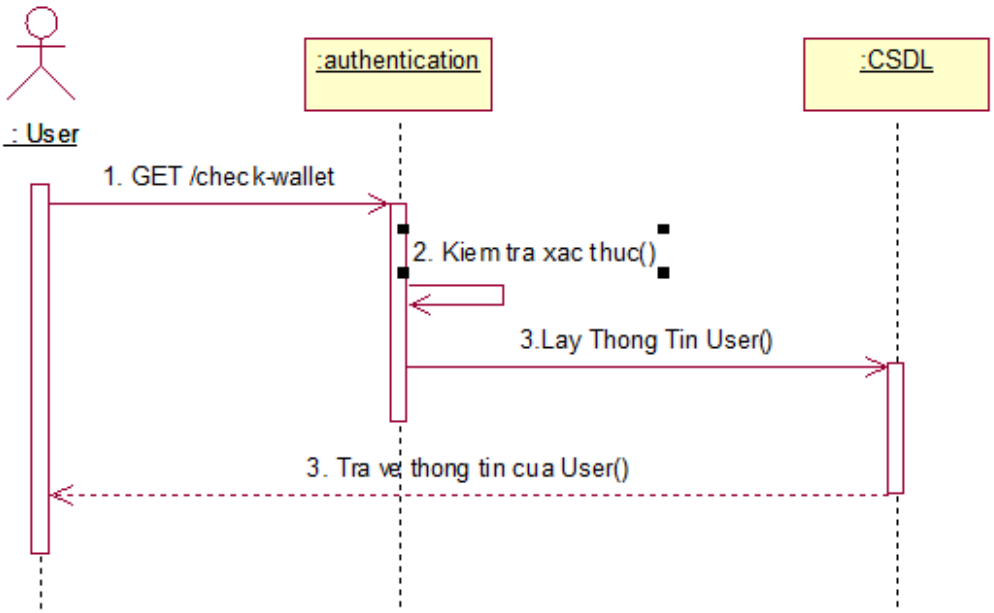
client_id	alphanumeric string	Thông tin tài khoản đăng nhập
currency	alphanumeric string (max. 3 character)	Đơn vị tiền tệ sử dụng, tiêu chuẩn ISO 4217, ví dụ USD, VND, …
billing_model	alphanumeric string	Phương thức thanh toán - Debit : trả trước - Credit : trả sau
balance	float (15.3)	Tổng số dư tài khoản của khách hàng - Số dư tài khoản (debit) - Số tiền ghi nợ (credit)
wallet	float (15.3)	- Số dư đầu kỳ (debit) - Giới hạn thanh toán (credit)

Ví dụ:

{ Request
URI: GET http://vta-address:vta-port/check-wallet

Response:

"code":"0",
"massage":"Successful Request",
"client_id":"tester",
"currency":"USD",
"billing_model":"Credit",
"balance":"15",
"wallet":"400"
}



Hình 4.5. Lược đồ tuần tự check wallet

4.5.3 Phương thức "Service Info"

Phương thức này được sử dụng để có được thông tin về dịch vụ hỗ trợ, số tiền topup và danh sách lệ phí kết nối đến công ty viễn thông mà khách hàng đã đăng ký. Hình 4.5 chỉ ra luồng khi người dùng cần lấy thông tin của các dịch vụ Topup có trên hệ thống. Để thực hiện hành động này máy khách sẽ gửi một yêu cầu HTTP GET tới URI /service-inft/{class}/{object} với {class} là định danh nhóm tài nguyên mà người dùng muốn nhận và {object} là đối tượng chi tiết hơn của những nhóm tài nguyên trên. Chi tiết các trường hợp của {class} và {object} được mô tả ở bảng dưới đây. Yêu cầu của khách hàng sẽ được hệ thống VTA tiếp nhận. Đầu tiên mô đun xác thực sẽ phân tích URI và các tham số header để xác thực người dùng như đã nêu trong phần 4.3.6. Trong trường hợp người dùng được xác thực hệ thống sẽ kiểm tra sự hợp lệ của các biến đầu vào. Nếu các biến đó thỏa mãn và đúng quy tắc hệ thống sẽ yêu cầu thông tin của dịch vụ trong CSDL và tạo ra một kết quả HTTP gửi lại cho người dùng.

Định dạng request

URI: GET http://vta-address:vta-port/service-info/{class}/{object}

Trường	Kiểu dữ liệu	Mô tả
class	alphanumeric string	Những quy định về các thông tin truy vấn: - "**countries**": trả về một danh sách của tất cả các nước có sẵn mà khách hàng đã đăng ký

		- "**country**" : trả về một danh sách của tất cả công ty viễn thông có sẵn mà khách hàng đã đăng ký - "**telco**" : trả về thông tin về các dịch vụ hỗ trợ Topup, số tiền và danh sách phí
object	alphanumeric string	- Nếu class = countries, khi đó object = **null** - Nếu class = country, khi đó object = "**country_code**": ISO 3166-1 alpha-2 code, ví dụ: US, VN, … - Nếu class = telco, khi đó object = "**telco_id"**: ID của requested telco

Định dạng response

Trường	Kiểu dữ liệu	Mô tả
Code	interger	Mã lỗi của request
Message	string	Chú thích cho mã lỗi.
country_code	alphanumeric string (max. 2 character)	Mã quốc gia của công ty viễn thông đích, tiêu chuẩn ISO 3166-1 alpha-2 code, ví dụ: VN, US,…
country_name	alphanumeric string	Tên quốc gia của công ty viễn thông đích đến
telco_id	interger	ID của công ty viễn thông được Topup
telco_name	alphanumeric string	Tên của công ty viễn thông được Topup
service_id	interger	ID của dịch vụ được Topup
service_name	alphanumeric string	Tên của dịch vụ được Topup
destination_currency	alphanumeric string	Đơn vị tiền tệ của quốc gia được Topup
destination_amounts	float (9.3)	Số tiền nhận bởi người nhận
source_currency	alphanumeric string	Đơn vị tiền tệ dùng với khách hàng
source_amounts	float (9.3)	Số tiền gửi của người gửi
fee_amounts	float (9.3)	Phí tính trên mỗi giá trị Topup
transmitted_amounts	float (9.3)	Số tiền thực tế gửi tới nhà mạng đích

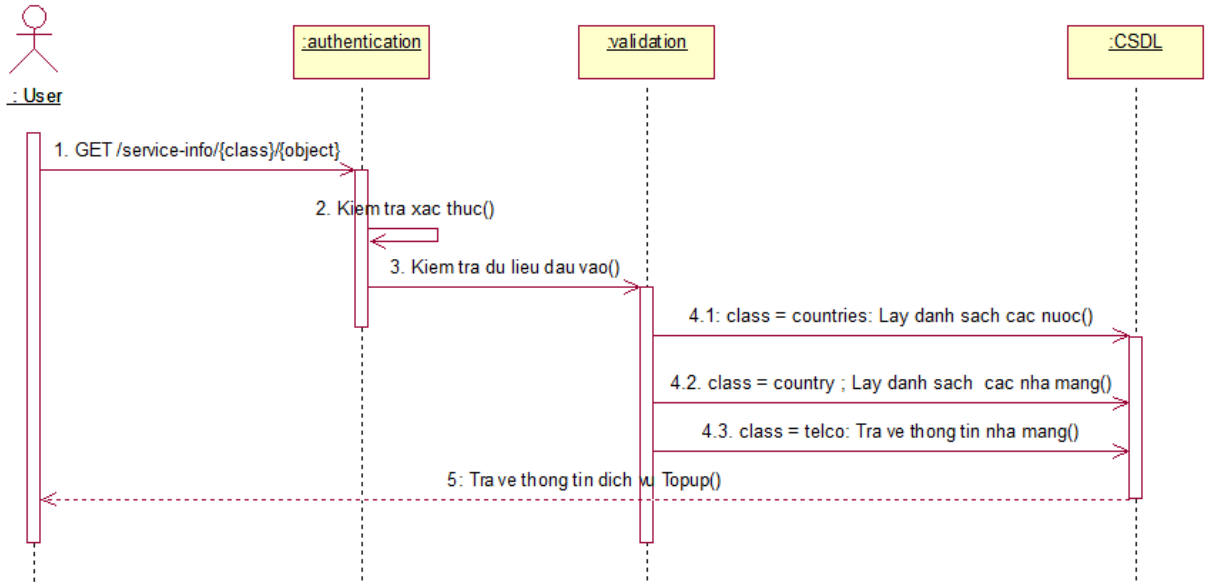
Ví dụ:

Request

URI: GET http://vta-address:vta-port/service-info/telco/2

Response:


```
{
"code":"0",
"massage":"Successful Request",
"country_code":"HT",
"country_name":"HAITI",
"telco_id":"2",
"telco_name":"Natcom",
"service":
  {
  "service_id":"1",
  "service_name":"Prepaid Mobile Recharge",
  "source_currency":"USD",
  "source_amounts":"5,6,7",
  "fee_amounts":"1.05,1.06,1.07",
  "transmitted_amounts":"3.95,4.94,5.93",
  "destination_currency":"HTG",
  "destination_amounts":"172,216,259"
  }
}
```



Hình 4.6. Lược đồ tuần tự lấy thông tin dịch vụ

4.5.4 Phương thức "Topup"

Phương pháp này được sử dụng để nạp một tài khoản đích cho một dịch vụ xác định với một khoản tiền nhất định. Hình 4.6 chỉ ra luồng khi người dùng cần thực hiện hành động TopUp. Để thực hiện hành động trên máy khách sẽ gửi một yêu cầu HTTP POST tới URI /topup bao gồm các tài nguyên để thực hiện hành động TopUp. Cũng như các trường hợp khác đầu tiên hệ thống sẽ xác thực người dùng thông qua các tham số URI, header và nội dung tài nguyên đính kèm. Tiếp theo hệ thống sẽ kiểm tra sự hợp lệ của các biến đầu vào. Khi dữ liệu đầu vào là hợp lệ hệ thống sẽ kiểm thông tin người dùng để đảm bảo rằng tài khoản người dùng là đủ để thực hiện giao dịch. Để thực hiện giao dịch hệ thống sẽ tính phí giao dịch dựa vào biểu mẫu có sẵn và thông tin giao dịch của khách hàng. Nếu các bước trên hoàn thành và không xảy ra lỗi hệ thống sẽ thực hiện giao dịch TopUp, thực hiện lưu log giao dịch, tạo ra ID giao dịch và tạo ra một thông báo về giao dịch là một kết quả HTTP gửi lại cho người dùng.

Định dạng request

URI: POST http://vta-address:vta-port/topup

Tham số truyền vào

Field	Kiểu dữ liệu	Mô tả
telco_id	interger	ID của công ty viễn thông của người nhận
service_id	interger	ID của dịch vụ cần phải nạp tiền
sender	alphanumeric	Số điện thoại quốc tế của người sử dụng

	string	muốn nạp tiền tài khoản của người nhận (Phải ở định dạng MSISDN*)
Recipient	alphanumeric string	Tài khoản để áp dụng thanh toán. Cần định dạng MSISDN hoặc tài khoản thanh toán
Amount	float (9.3)	Số tiền áp dụng cho người nhận (bằng tiền của khách hàng)
client_transaction_id	alphanumeric string	ID giao dịch được tạo ra bởi hệ thống khách hàng, là duy nhất cho mỗi giao dịch (chúng tôi sẽ hỗ trợ tính năng gửi giao dịch không thành công trong tương lai)

(*) MSISDN được xây dựng bằng: ***MSISDN = CC + NDC + SN***

- *CC = Mã quốc gia*
- *NDC = Mã điểm đến quốc gia*
- *SN = Số thuê bao*

Ví dụ:

- *509987654321 = 509 – – 987654321,*
- *84444501982 = 84 – 4 – 44501982*

Định dạng response

Trường	Kiểu dữ liệu	Mô tả
Code	interger	Mã lỗi của request
Message	String	Chú thích của mã lỗi
vta_transaction_id	alphanumeric string	ID giao dịch của hệ thống VTA
source_currency	alphanumeric string	Đơn vị tiền tệ của khách hàng thực hiện các yêu cầu Topup
source_amount	float (9.3)	Số tiền Topup theo yêu cầu của người gửi
fee_amount	float (9.3)	Phí tính trên số tiền Topup
transmitted_amount	float (9.3)	Số tiền thực tế gửi đến nhà mạng đích
destination_currency	alphanumeric string	Đơn vị tiền tệ của quốc gia đích của
destination_amount	float (9.3)	Số tiền Topup nhận bởi người nhận

trans_time	datetime	Ngày và giờ của giao dịch (YYYY-MM-DD hh:mm:ss)

Ví dụ:

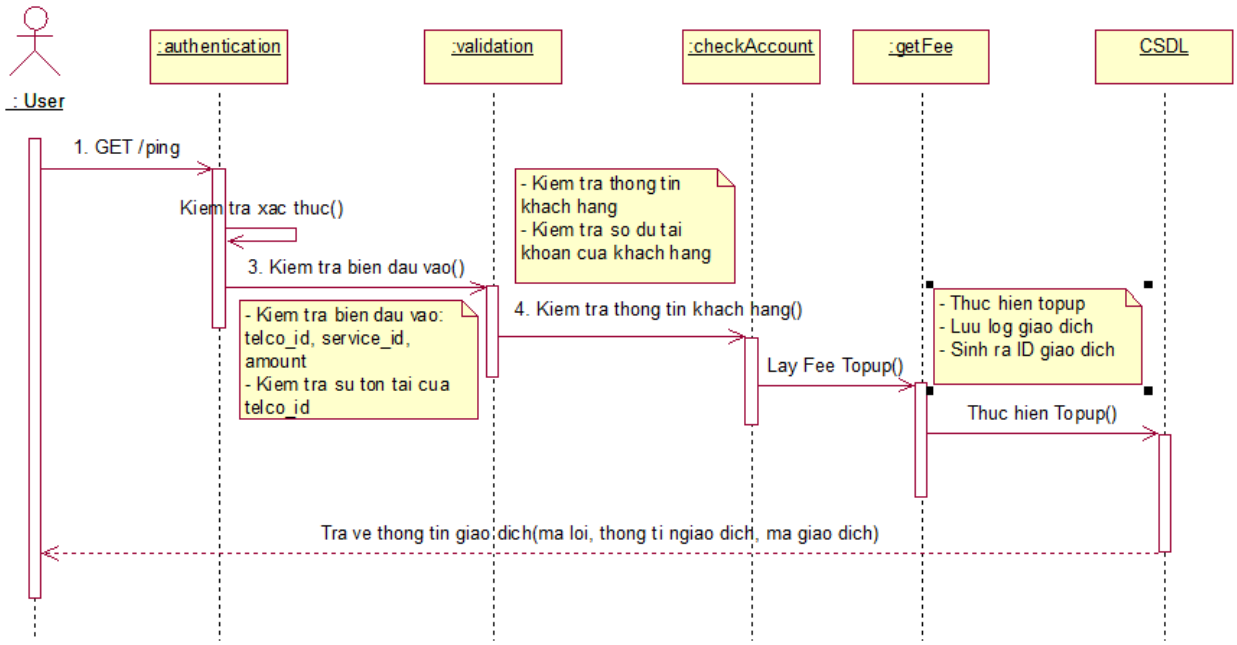
URI: POST http://vta-address:vta-port/topup

Tham số truyền vào


```
{
"telco_id":"2",
"service_id":"1",
"sender":"12356789",
"recipient":"50943746892",
"amount":"5",
"currency":"USD",
"client_transaction_id":"131313"
}
```


Response:


```
{
"code":"0",
"message":"Transaction Successful",
"country":"HT",
"vta_transaction_id":"234",
"source_currency":"USD",
"source_amount":"5",
"fee_amount":"1.05",
"transmitted_amount":"3.95",
"destination_currency":"HTG",
"destination_amount":"172",
"trans_time":"2014-01-15 00:23:14"
}
```



Hình 4.7. Lược đồ tuần tự hành động TOPUP

4.5.5 *Phương thức "Trans History"*

Đây là phương thức truy vấn lịch sử giao dịch Topup. Hình 4.7 chỉ ra luồng khi người dùng cần tra cứu lịch sử giao dịch trên hệ thống. Để thực hiện hành động trên máy khách sẽ gửi một yêu cầu HTTP GET tới URI /trans-history/{ vta_transaction_id } với {vta_trasaction_id} là mã giao dịch xác định lịch sử giao dịch cần lấy. Yêu cầu được hệ thống VTA tiếp nhận, đầu tiên mô đun xác thực sẽ phân tích URI và các tham số header để xác thực người dùng như đã nêu trong phần 4.3.6. Trong trường hợp người dùng được xác thực hệ thống sẽ kiểm tra sự hợp lệ của các biến đầu vào. Nếu các biến đầu vào thỏa mãn tất cả các yêu cầu của hệ thống khi đó hệ thống sẽ yêu cầu thông tin từ CSDL về giao dịch theo mã giao dịch mà khách hàng đã gửi đến. Và cuối cùng hệ thống tạo ra một kết quả HTTP gửi lại cho người dùng.

Định dạng request

URI: GET http://vta-address:vta-port/trans-history/{ vta_transaction_id }

Định dạng response

Trường	Kiểu dữ liệu	Mô tả
Code	Interger	Mã lỗi của request
Message	String	Chú thích mã lỗi
client_transaction_id	alphanumeric string	ID giao dịch của hệ thống khách hàng

vta_transaction_id	alphanumeric string	ID giao dịch của hệ thống VTA
history_code	Interger	Mã lỗi của giao dịch được truy vấn
history_message	String	Chú thích mã lỗi của giao dịch được truy vấn
source_currency	alphanumeric string	Đơn vị tiền tệ của khách hàng thực hiện Topup
source_amount	Numeric	Số tiền Topup theo yêu cầu của người gửi
fee_amount	Numeric	Phí tính trên số tiền Topup
transmitted_amount	float (9.3)	Số tiền thực gửi đến công ty viễn thông đích
destination_currency	alphanumeric string	Đơn vị tiền tệ của quốc gia đích của người nhận
destination_amount	Numeric	Số tiền Topup nhận bởi người nhận
trans_time	Datetime	Ngày và thời điểm giao dịch (YYYY-MM-DD hh:mm:ss)

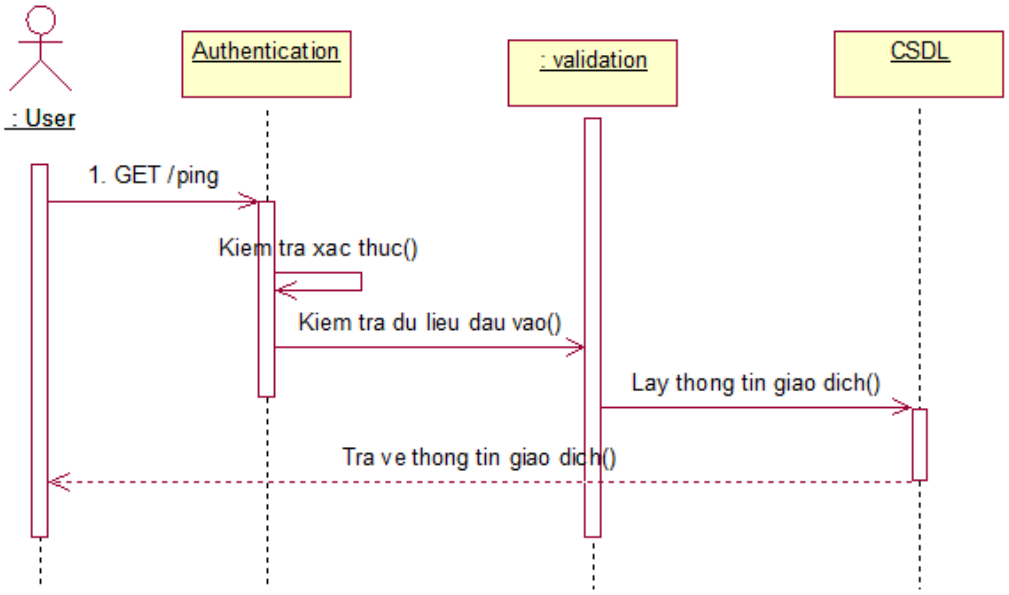
Ví dụ:

Request:

URI: GET http://vta-address:vta-port/trans-history/234

Response:


```
{
"code":"0",
"message":"Successful Request",
"client_transaction_id":"131313",
"vta_transaction_id":"234",
"history_code":"0",
"history_message":"Transaction Successful",
"source_currency":"USD",
"source_amount":"5",
"fee_amount":"1.05",
"transmitted_amount":3.95,
"destination_currency":"HTG",
"destination_amount":"172",
"trans_time":"2014-01-15 00:23:14"
}
```



Hình 4.8. Lược đồ tuần tự hành động lấy lịch sử giao dịch

4.5.6 Danh sách mã lỗi

Đây là danh sách đầy đủ các mã lỗi trả về bởi hệ thống VTA Topup:

Code	Tin nhắn	HTTP Code	Ghi chú
0	Transaction Successful/ Successful Request	200	
10	Request is wrong format or not enough field	401	
11	Client-ID not found or be blocked	401	
12	Client IP address is malformed or not allowed	401	
13	Invalid Format Timestamp	403	
14	Invalid Request Timestamp	403	
15	Duplicate Request Timestamp	403	
16	Can't verify signature	403	
17	Request denied. Please contact support	403	
40	Invalid input data	400	Missing required input

			data
41	Invalid service class value	400	Only support value: “countries”, “country”, “telco”
42	Invalid service object value	400	Input valid country_code, telco_id
43	Your account is not allowed to use this service	400	
49	Invalid action value	400	
51	Duplicate client_transaction_id	400	
52	Invalid format or Out of range topup amount	400	
56	Transaction ID not found	400	
66	The recipient number is malformed or not exist	200	
67	The recipient number is a pos-paid subscriber, transaction refused	200	
68	Transaction Incomplete	200	
69	Request Timeout	200	
80	Your credit limit is exceeded. Can not perform topup action	200	
98	System not available. Please try again later	500	
99	Unexpected Error. Please contact support	500	

4.6 Thử nghiệm và đánh giá kết quả

4.6.1 Giới thiệu

Các khái niệm kỹ thuật của REST khá là trừu tượng về mặt lý thuyết, vì vậy để hiểu rõ về REST tác giả đã xây dựng một ứng dụng mô phỏng các API được nêu trong luận văn này. Ứng dụng được xây dựng dựa trên framework Laravel. Máy chủ web dùng để chạy ứng dụng là Xampp. Xampp là chương trình tạo máy chủ Web được tích hợp sẵn Apache, PHP, MySQL, FTP Server, Mail Server và các công cụ như

phpMyAdmin. Xampp có chương trình quản lý khá tiện lợi, cho phép chủ động bật tắt hoặc khởi động lại các dịch vụ máy chủ bất kỳ lúc nào.

4.6.2 Một số đoạn code mô tả thực thi API

Lấy thông tin từ Header

Hình 4.9 chỉ ra đoạn code lấy thông tin xác thực từ Header của các API. Các thông tin được truyền vào Header gồm: client_id, timestamp, signature được dùng xác thực và bảo mật API.

```
$input['client_id'] = Request::header('Client-ID');
$input['timestamp'] = Request::header('Timestamp');
$input['signature'] = Request::header('Signature');
```

Hình 4.9 Lấy thông tin được truyền vào Header

Hàm xác thực và tạo chữ ký

Hình 4.10 chỉ ra đoạn code của phương thức xác thực và tạo chữ ký. Chữ ký được tạo ra bằng các áp dụng JSON Web Token. Khóa công khai ở đây được tạo ra từ client_id, timestamp và các biến truyền vào API, khóa riêng là một mã bí mật được tạo ra lúc tạo client_id. Việc so sánh chữ ký được truyền vào HTTP Header và chữ ký tạo trên API sẽ xác định thông tin truyền vào có đáng tin cậy hay không.

```
public function authentication($input){
    $key = $this->key;
    $sig = $this->createSignature($input['client_id'].$input['timestamp'].$input['body'],$key);
    if($sig == $input['signature']){
        return 1;
    }else{
        return 0;
    }

}
public function createSignature($message,$key){
    return  base64_encode(hash_hmac('sha1', $message, $key));
}
```

Hình 4.10 Phương thức xác thực và tạo chữ ký

4.6.3 Dùng thử API

Để kiểm tra trực tiếp các API được tạo ra tác giả đã xây dựng một ứng dụng mô phỏng việc sử dụng API.

Một số đoạn code mô tả dùng thử API

Hình ảnh 4.11 mô tả việc tạo một mảng request_header. Mảng này sẽ được truyền vào HTTP Header lúc gọi API.

```
$request_headers = array();
$request_headers[] = 'Client-ID:'.$client_id;
$request_headers[] = 'Timestamp: '.$timestamp;
$signature = $this->createSignature($client_id.$timestamp.$input,$key);
$request_headers[] = 'Signature:'.$signature;
```

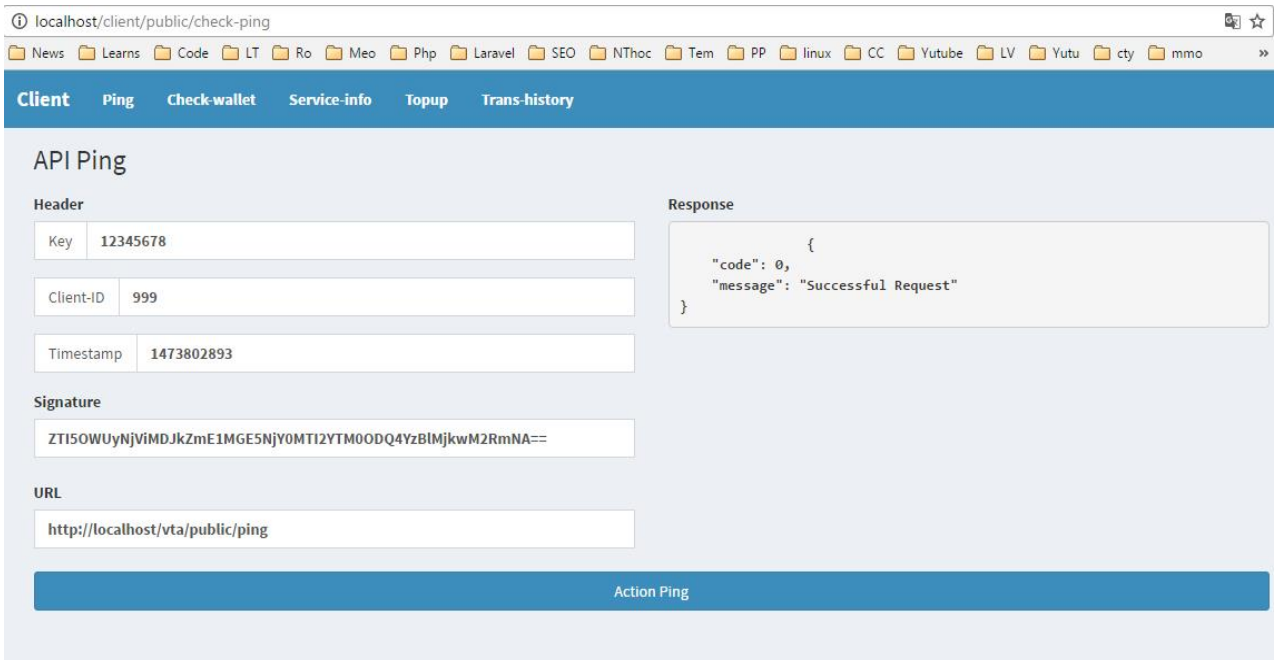
Hình 4.11 Hình ảnh tạo mảng request_header

Hình 4.12 mô tả việc gọi API Ping bằng cách sử dụng hàm CURL

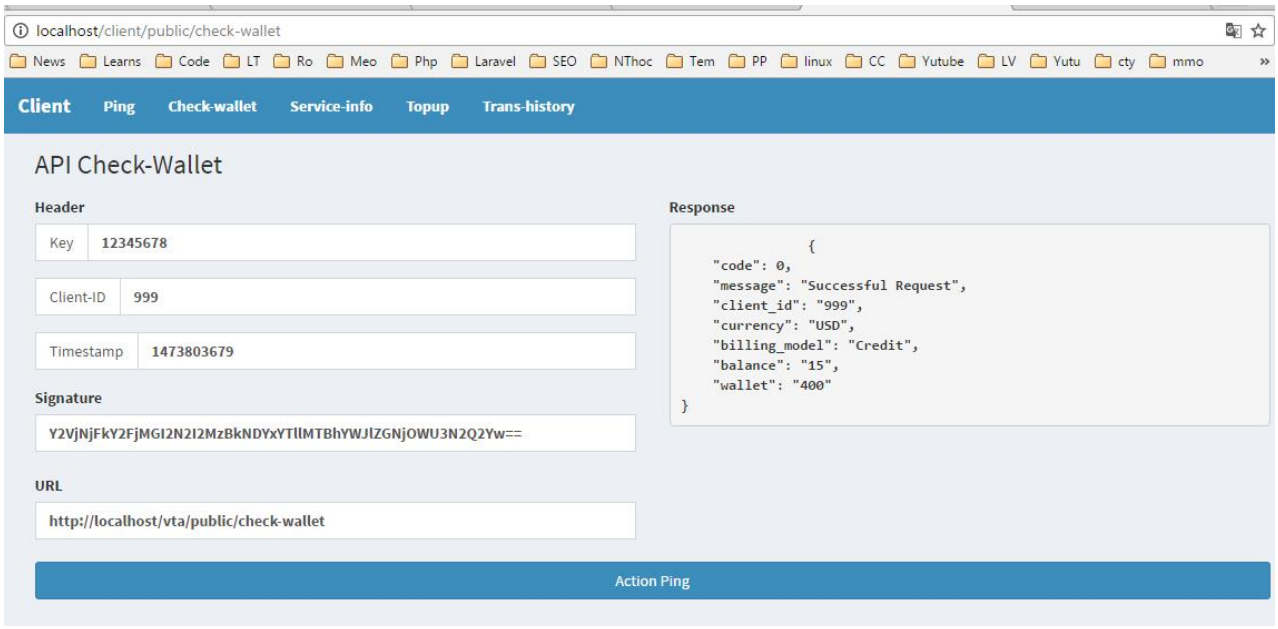
```
$url = "http://localhost/vta/public/ping";
$ch = curl_init($url);
curl_setopt($ch,CURLOPT_HTTPHEADER,$request_headers);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
$op=curl_exec($ch);
curl_close($ch);
```

Hình 4.12 Hình ảnh mô tả việc gọi api ping

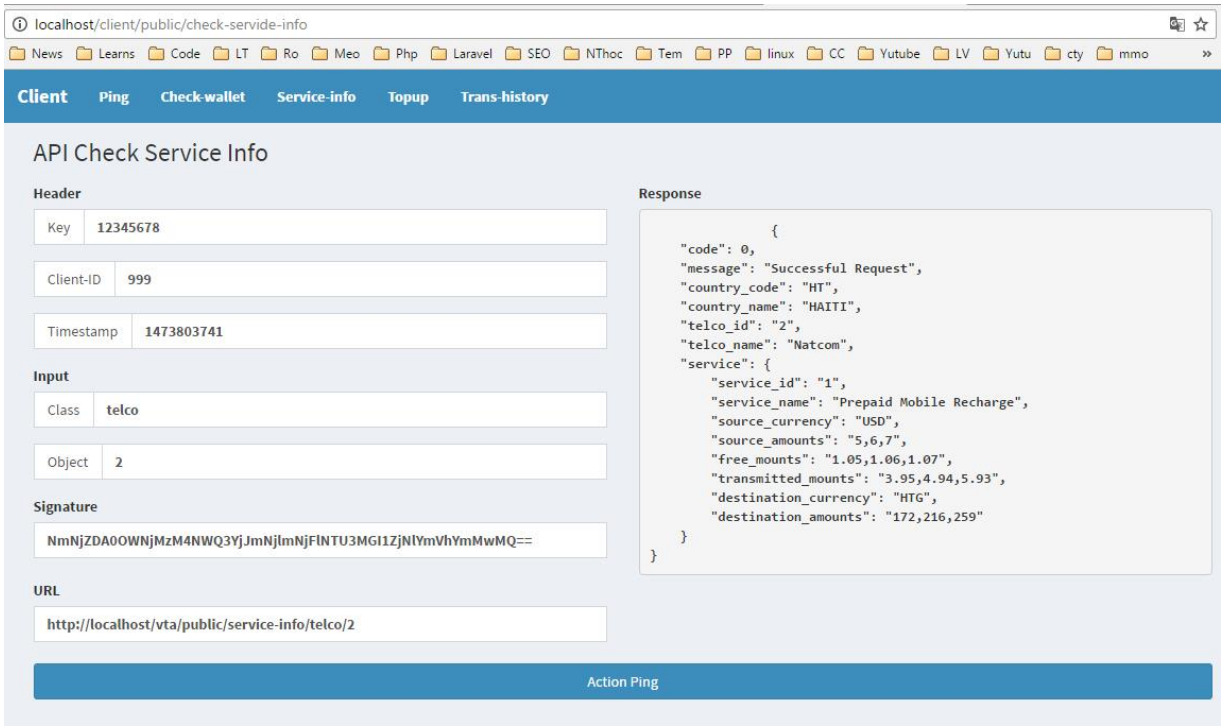
Dùng thử API Ping



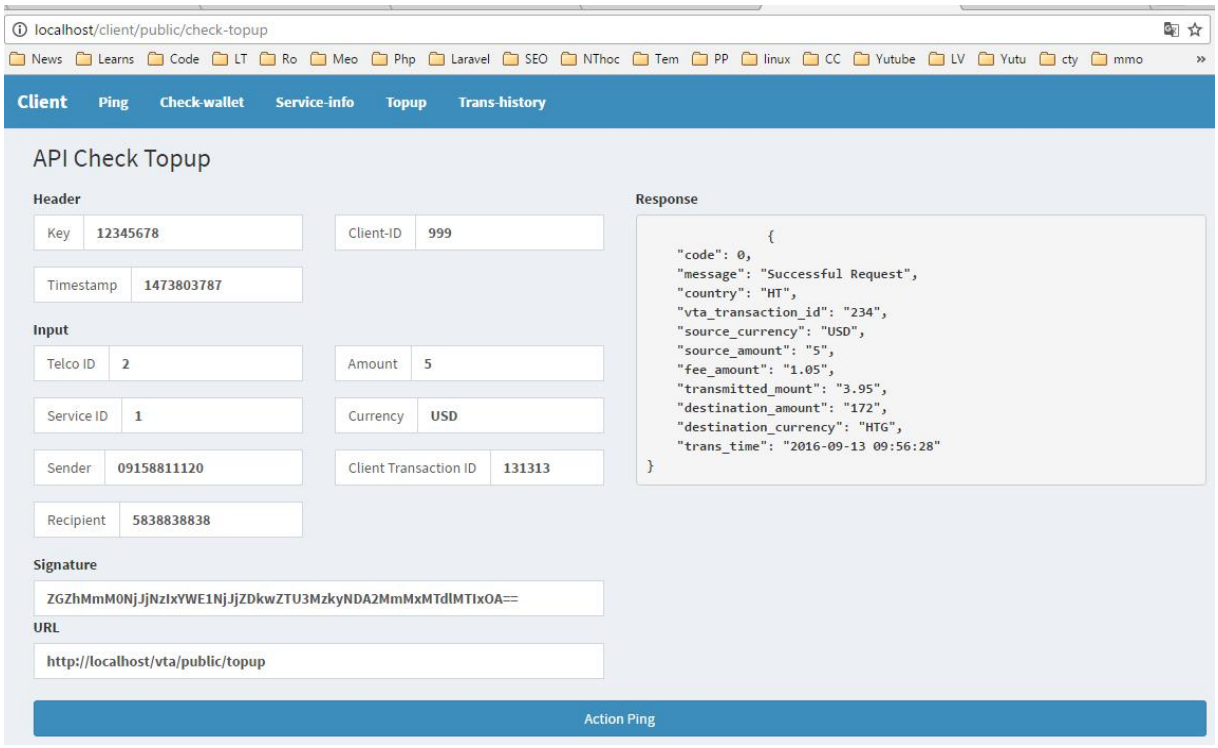
Hình 4.13 Hình ảnh giao diện dùng thử API Ping



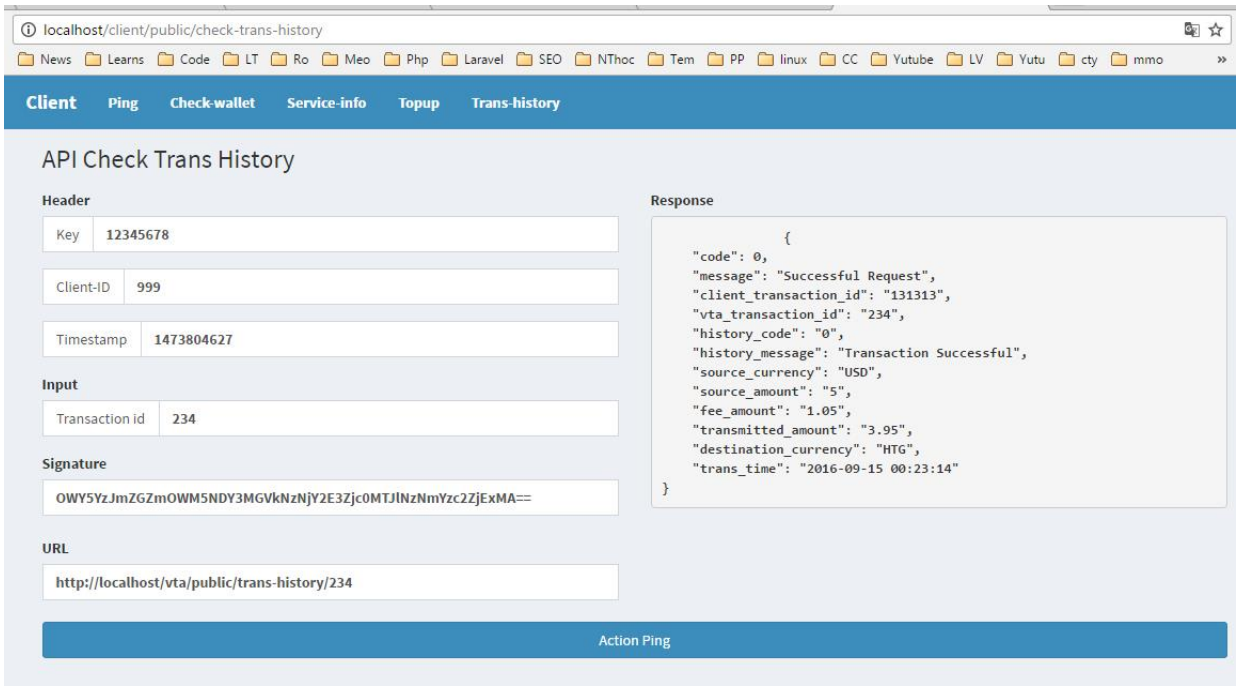
Hình 4.14 Hình ảnh giao diện dùng thử API Check Wallet



Hình 4.15 Hình ảnh giao diện dùng thử API Service Info



Hình 4.16 Hình ảnh giao diện dùng thử API Topup



Hình 4.17 Hình ảnh giao diện dùng thử API Trans History

KẾT LUẬN

1. Kết luận

Hầu hết các web API ngày nay có thể được truy cập từ bất kỳ nơi đâu trên thế giới thông qua web. Trong dự án này, dịch vụ web kiểu REST được thực thi dùng thiết kế RESTful API, và kết quả nhận được thật sự thỏa đáng. Do đó có thể nói rằng hệ thống API có nhiều yêu cầu phức tạp tuy nhiên có thể dễ dàng thực hiện với REST.

Cùng với sự tham gia của hai thành viên Nguyễn Vũ Hà và Hoàng Việt Long, chúng tôi đã xây dựng thành công hệ thống API TOPUP. Thành viên Nguyễn Vũ Hà có vai trò xây dựng phần bảo mật và xác thực cho hệ thống API. Thành viên Hoàng Việt Long có vai trò xây dựng các Test Case cho hệ thống API. Và tác giả chịu trách nhiệm xây dựng bộ API với kiến trúc REST và phối hợp với hai thành viên còn lại để kiểm tra các lỗi hệ thống và khắc phục những lỗi phát sinh trong quá trình xây dựng.

Qua quá trình tìm hiểu và thiết kế REST API, tác giả có thể nói rằng kiểu kiến trúc REST là một giải pháp thích hợp đối với hệ thống API. REST có thể thay thế XML-PRC API và GUI API với một REST API chung nhất và duy nhất mà vẫn có thể điều khiển được các dịch vụ. REST API đã được kiểm chứng rằng cả máy tính và con người sử dụng REST mà dữ liệu trả về theo nhiều kiểu đại diện khác nhau. Hơn nữa với một API chung thực sự rõ ràng hơn, rành mạch hơn và có thể sử dụng lại API này cho các hệ thống web khác khi REST là chung thực sự. REST cũng đã chứng tỏ được rằng ngay khi tìm hiểu được các khái niệm chính của REST thì nhiệm vụ thiết kế một API theo các nguyên tắc đó không quá phức tạp như ban đầu mới tiếp xúc REST.

- Những nhiệm vụ của đề tài đã hoàn thành:
    + Tìm hiểu và ứng dụng REST
    + Tìm hiểu khung làm việc Laravel
    + Xây dựng thử nghiệm hệ thống Topup sử dụng API theo nguyên tắc REST.
- Hạn chế:
    + Hiện tại hệ thống Topup mới chỉ đáp ứng kết nối các kênh thanh toán / topup trực tiếp của người dùng.

2. Hướng phát triển

- Xây dựng các Wholesale WS là Gateway đáp ứng kết nối các kênh thanh toán/ topup của các đối tác bán buôn cước viễn thông.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] Topup - http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/TOPUP

[2] R. Fielding et al (1999). Hypertext Transfer Protocol – HTTP/1.1. IETF RFC 2616.

[3] T. Berners-Lee et al (1996). Hypertext Transfer Protocol – HTTP/1.0. IETF RFC 1945.

[4] R. Fielding et al (1997). Hypertext Transfer Protocol – HTTP/1.1. IETF RFC 2068.

[5] R. Fielding, editor (2006). RFC for REST. REST Discussion Mailing List.

[6] R. Fielding (2000). Architectural Styles and The Design of Network-based Software Architectures. PhD thesis, University of California, Irvine.

[7] L. Richardson, S. Ruby, et al (2007). Restful Web Services. O‟Reilly, 1st edition.

[8] World wide web consortium (2004). http://www.w3.org/. W3C.

[9] Laravel Framework. http://laravel.com/.

[10] M. Gudgin et al (2007). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation. W3C.

[11] E. Christensen et al (2001). Web Services Description Language (WSDL) 1.1. W3C Note. W3C.

[12] C. Pautasso, O. Zimmermann, and F. Leymann (2008). RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. IW3C2.

[13] Restful webservices (2008). http://www.slideshare.net/gouthamrv/restful-services-2477903.

[14] P. James. Http caching (2006). http://www.peej.co.uk/articles/http-caching.html.

[15] Nadia Mohedano Troyano (2010). The Design of a RESTful Web Service. PhD thesis, kungliga tekniska hÖgskolan school of electrical engineering tnssm.