**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**

**----------\*\*\*----------**

****

**BÁO CÁO PROJECT II**

**Đề tài: Xây dựng topology mạng**

|  |  |
| --- | --- |
| Giảng viên hướng dẫn: | ThS. Bùi Trọng Tùng |
| Sinh viên: | Nguyễn Tùng Lâm |

MỤC LỤC

[**CHƯƠNG I: TỔNG QUAN** 3](#_Toc517258905)

[**CHƯƠNG II: WEB SERVICE TẠI SERVER** 4](#_Toc517258906)

[**I.** **Quản lý danh sách topology** 4](#_Toc517258907)

[**I.1.** **Vấn đề** 4](#_Toc517258908)

[**I.2.** **Xây dựng** 4](#_Toc517258909)

[**II.** **Xây dựng topology** 6](#_Toc517258910)

[**II.1.** **Đồng bộ dữ liệu giữa server và proxy** 6](#_Toc517258911)

[**II.2.** **Tổng hợp dữ liệu** 10](#_Toc517258912)

[**CHƯƠNG III: MODULE JS QUẢN LÝ VÀ HIỂN THỊ TOPOLOGY** 13](#_Toc517258913)

[**I.** **Quản lý danh sách topology** 13](#_Toc517258914)

[**I.1.** **Các chức năng** 13](#_Toc517258915)

[**I.2.** **Phương thích tích hợp** 15](#_Toc517258916)

[**II.** **Hiển thị và thao tác với topology** 16](#_Toc517258917)

[**II.1.** **Hiển thị topology** 16](#_Toc517258918)

[**II.2.** **Các thao tác với topology** 18](#_Toc517258919)

[**CHƯƠNG IV: KẾT QUẢ ĐẠT ĐƯỢC VÀ PHƯƠNG HƯỚNG PHÁT TRIỂN** 28](#_Toc517258920)

[**I.** **Kết quả đạt được** 28](#_Toc517258921)

[**I.1.** **Web service tại server** 28](#_Toc517258922)

[**I.2.** **Module JS quản lý và hiển thị topology** 28](#_Toc517258923)

[**II.** **Phương hướng phát triển** 28](#_Toc517258924)

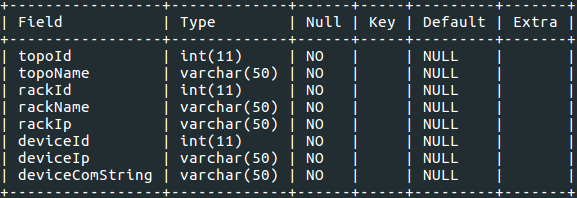
# **CHƯƠNG I: TỔNG QUAN**

* **Yêu cầu**
* Xây dựng giao diện hiển thị để người quản trị mạng xem và quản lý topology mạng của một hệ thống mạng cụ thể.
* **Phân tích**
* Mỗi rack controller có một zabbix proxy quản lý. Zabbix proxy lấy thông tin về các thiết bị trong mạng của mình bằng giao thức SNMP và lưu trữ trong cơ sở dữ liệu MySQL. Đồng thời tại mỗi proxy xây dựng một web service để thực hiện lấy thông tin thiết bị khi có lệnh từ server và cung cấp nội dung các bảng.
* Zabbix server quản lý các proxy, có thể gửi lệnh kèm danh sách IP và community string tương ứng để proxy quét lấy thông tin, gọi đến web service của proxy để đồng bộ thông tin từ cơ sở dữ liệu của proxy lên server phục vụ xây dựng topology mạng.
* Web service trên server cung cấp các chức năng giao tiếp với proxy, tổng hợp thông tin từ các rack controller và lưu trữ lại trong cơ sở dữ liệu MySQL. Đồng thời cung cấp các dữ liệu tổng quan về topology mạng để phục vụ quá trình hiển thị cho người dùng và thông tin chi tiết về các thiết bị để phục vụ cho quá trình quản lý giám sát thiết bị mạng.
* Server cung cấp thêm các chức năng cho phép người quản trị thêm, xóa, sửa các thiết bị mới vào một topology có sẵn để phục vụ quá trình quản lý, tuy nhiên các thiết bị được tổng hợp thông tin từ proxy sẽ không thể chỉnh sửa.
* Giao diện front-end hiển thị sơ đồ các thiết bị và liên kết giữa các thiết bị, cho phép xem thông tin chi tiết về các thiết bị, các cổng và trạng thái hoạt động của cổng đó thông qua môi trường làm việc trực quan, thân thiện, dễ sử dụng cho người quản trị thao tác.
* **Thiết kế**
* Web service trên zabbix proxy và server không yêu cầu giao diện, chỉ cung cấp các API để phục vụ lấy thông tin và thực hiện các tác vụ, do vậy được xây dựng bằng framework Flask của Python, giao tiếp với cơ sở dữ liệu MySQL bằng modul MySQLdb, dữ liệu trả về khi gọi API ở định dạng JSON.
* Giao diện front-end được xây dựng chủ yếu bằng thư viện đồ họa visjs và framework AngularJS, sử dụng AJAX request đến API của server lấy dữ liệu JSON để thực hiện các chức năng. Bên cạnh đó sử dụng thêm các thư viện ngDialog, ngDraggable để hiển thị các thông tin và các tác vụ kéo thả.

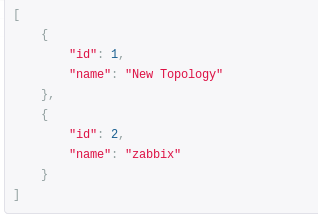
# **CHƯƠNG II: WEB SERVICE TẠI SERVER**

1. **Quản lý danh sách topology**
   1. **Vấn đề**

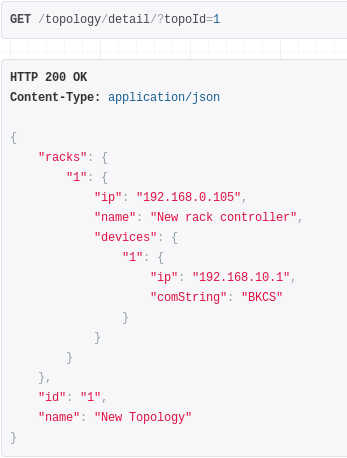
* Quản trị viên muốn quản lý topology của một hệ thống mạng cần cung cấp thông tin về các rack controller có trong mạng, mỗi rack controller cần thông tin về địa chỉ IP của proxy quản lý, địa chỉ IP của các thiết bị có trong rack controller đó và community string tương ứng với các thiết bị để proxy có thể tiến hành lấy thông tin bằng giao thức SNMP.
* Quản trị viên có thể tạo ra nhiều topology khác nhau, mỗi topology bao gồm một số rack controller cụ thể để có thể dễ dàng kiểm soát, tránh trường hợp dư thừa không cần thiết.
* Bên cạnh đó quản trị viên có thể xem và cập nhật thông tin về topology đã tạo trong trường hợp thông tin cấu hình đã bị thay đổi, cũng như có thể xóa một topology không cần sử dụng nữa.
  1. **Xây dựng**
* **Thiết kế cơ sở dữ liệu**
  + Bàng topoInfo:



* + Mỗi hàng trong bảng là một thiết bị với community string ứng với một rack controller trong một topology cụ thể.
* **Các chức năng**
  + Lấy danh sách các topology hiện có:
    - Đường dẫn: /topology/list/
    - Phương thức: GET
    - Tham số: Không.
    - Trả về: id và tên của các topology có trong bảng topoInfo.



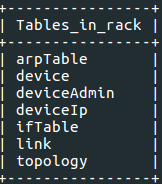
* + - Cài đặt: Lấy các trường topoId, topoName trong bảng topoInfo và định dạng lại dưới dạng JSON.
  + Lấy thông tin về một topology cụ thể:
    - Đường dẫn: /topology/detail/
    - Phương thức: GET
    - Tham số: topoId(GET): id của topology trong bảng topoInfo.
    - Trả về: Thông tin topology, các rack controller trong topology và các thiết bị trong mỗi rack.



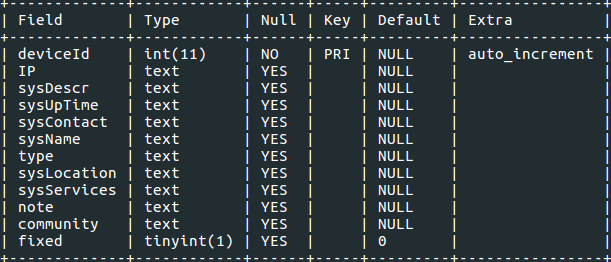
* + - Cài đặt: Lấy các trường của bảng topoInfo ứng với một topoId cụ thể ở biến topoId và định dạng JSON.
  + Thêm 1 topology
    - Đường dẫn: /topology/add/
    - Phương thức: POST
    - Tham số: data(POST): thông tin về một topology mới bao gồm tên, tên và địa chỉ proxy của các rack controller và các thiết bị cùng với community string trong các rack đó.
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Lấy id của topology có id cao nhất trong cơ sở dữ liệu để tính topoId của topology mới, tính rackId của các rack controller theo thứ tự tăng dần từ 1, deviceId của các device theo tứ tự tăng dần từ 1. Lần lượt đẩy các thiết bị với rackId và topoId vào bảng.
  + Chỉnh sửa 1 topology
    - Đường dẫn: /topology/edit/
    - Phương thức: POST
    - Tham số: data(POST): thông tin đã chỉnh sửa của một topology
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Lấy topoId từ tham số data, cập nhật rackIp, deviceIp và community string theo dữ liệu mới từ data.
  + Xóa 1 topology
    - Đường dẫn: /topology/delete/
    - Phương thức: POST
    - Tham số: data(POST): thông tin về topology cần xóa
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Lấy topoId từ tham số data, xóa tất cả các bản ghi tương ứng với topoId đó. Sau đó đặt lại topoId của các topology khác để đảm bảo luôn tăng liên tục từ 1.

1. **Xây dựng topology**
   1. **Đồng bộ dữ liệu giữa server và proxy**

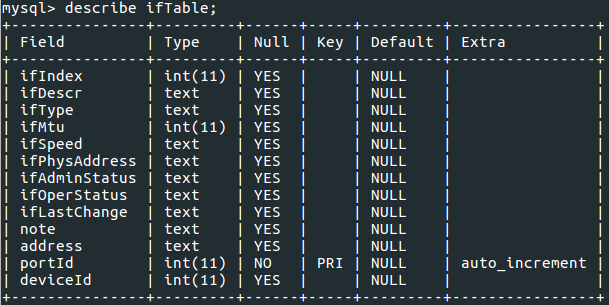
* **Cấu trúc cơ sở dữ liệu tại proxy**
* Cơ sở dữ liệu tại proxy quản lý một rack controller bao gồm thông tin về các thiết bị trong rack controller đó và liên kết giữa các thiết bị qua các cổng cụ thể. Cấu trúc các bảng trong cơ sở dữ liệu của proxy như sau:



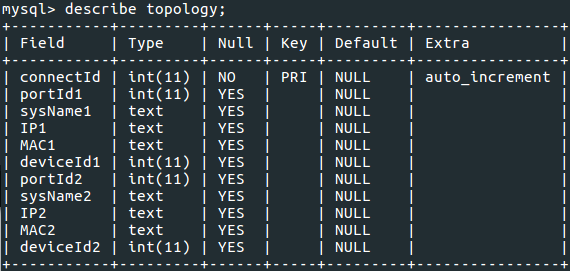
* Trong đó các bảng device, ifTable và topology là cần thiết để xây dựng topology và sẽ được đồng bộ lên server.
* Bảng device:



* Bảng ifTable:



* Bảng topology:



* Web service tại proxy cung cấp API cho phép lấy dữ liệu từ các bảng trong cơ sở dữ liệu và trả về dưới dạng JSON:

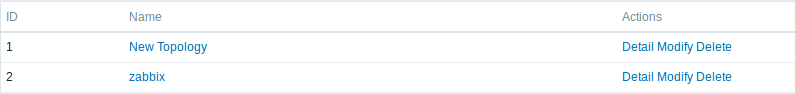


* **Lấy thông tin về các thiết bị tại rack controller**
  + Proxy quản lý rack controller sẽ nhận danh sách các thiết bị với IP và community string tại rack đó từ server để dùng snmp thu thập dữ liệu, sau đó lưu vào cơ sở dữ liệu mysql.
  + Server dựa vào các IP của proxy quản lý rack controller đã được cung cấp trong bảng topoInfo ứng với một topology mà cung cấp cho proxy danh sách thiết bị để proxy tiến hành lấy thông tin.
  + Chi tiết:
    - Đường dẫn: /general/discover/
    - Phương thức: GET
    - Tham số: TopoId(GET): Id của topology
    - Trả về: Thông báo thành công hoặc thất bại.
    - Cài đặt: Server lấy IP của các proxy quản lý các rack controller và danh sách các thiết bị tại rack đó trong bảng topoInfo được khởi tạo khi tạo một topology mới. Sau đó dùng IP để truy cập đến web service tại mỗi proxy và gửi kèm danh sách thiết bị cho proxy đó.
* **Đồng bộ dữ liệu**
  + Server dựa vào các IP của proxy quản lý rack controller đã được cung cấp trong bảng topoInfo ứng với một topology mà truy cập đến webservice của các proxy đó để lấy nội dung các bảng, sau đó tổng hợp lại trong cơ sở dữ liệu của server cộng thêm thông tin về topoId và rackId.
  + Dữ liệu lấy được từ các rack controller sẽ được so sánh với dữ liệu cũ trong cơ sở dữ liệu của server, nếu đã tồn tại thì cập nhật, nếu chưa tồn tại thì đẩy vào, sau đó xóa các dữ liệu cũ đã lỗi thời đi nhưng giữ nguyên dữ liệu do người dụng tự thêm vào.
  + Chi tiết:
    - Đường dẫn: /general/synchronize/
    - Phương thức: GET
    - Tham số: topoId(GET): Id của topology cần đồng bộ dữ liệu
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Đồng bộ lần lượt theo các bảng, bắt đầu từ bảng Device, sau đó đến bảng IfTable và cuối cùng là bảng Topology. Với mỗi bảng đều kiểm tra sự tồn tại của dữ liệu cũ để quyết định cập nhật hoặc thêm mới. Các trường thêm vào của các bảng là topoId, rackId và fixed. Trường fixed quy định dữ liệu là đồng bộ từ topology (mang giá trị 1) hay do người quản trị bổ sung thủ công (mang giá trị 0). Các bản ghi mới được đồng bộ sẽ mang giá trị fixed=2, sau đó xóa tất cả các bản ghi cũ có giá trị fixed=1 và cập nhật lại biến fixed=1 cho các bản ghi vừa được đồng bộ.
  1. **Tổng hợp dữ liệu**
* **Dữ liệu chung**
  + Lấy dữ liệu về một topology cụ thể:
    - Đường dẫn: /general/detail/
    - Phương thức: GET
    - Tham số: topoId(GET): ID của topology cần lấy dữ liệu
    - Trả về: Dữ liệu của topology dưới định dạng JSON hoặc thông báo lỗi nếu có lỗi xảy ra.
    - Cài đặt: Lấy danh sách các thiết bị trong topology tại bảng device để xây dựng danh sách nút. Lấy liên kết giữa các cổng từ bảng topology, ứng với mỗi cổng lại so khớp trong bảng IfTable để tìm thiết bị tương ứng, từ đó xây dựng cạnh giữa các nút cùng số hiệu cổng tại các nút ứng với cạnh đó.
  + Lưu thay đổi:
    - Đường dẫn: /general/save/
    - Phương thức: GET
    - Tham số: Không
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Commit những thay đổi thủ công của người quản trị vào topology. Những thay đổi như thêm, xóa, sửa nút của topology sẽ không trực tiếp thay đổi trong cơ sở dữ liệu và sẽ mất khi kết thúc phiên, do vậy cần gọi chức năng này để lưu những thay đổi muốn giữ lại.
  + Hủy thay đổi:
    - Đường dẫn: /general/discard/
    - Phương thức: GET
    - Tham số: Không
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Discard những thay đổi thủ công của người quản trị vào topology. Nếu người quản trị nhỡ tay thêm hoặc xóa nhầm thì có thể discard những thay đổi đó về lần save gần nhất.
* **Nút** 
  + Lấy thông tin chi tiết về nút:
    - Đường dẫn: /node/detail
    - Phương thức: GET
    - Tham số: topoId(GET): Id của topology, deviceId(GET): id của thiết bị cần lấy thông tin trong topology đó.
    - Trả về: Thông tin chi tiết về một nút hoặc thông báo lỗi
    - Cài đặt: Lấy tất cả thông tin trong bảng device về thiết bị với deviceId trong topology được chỉ định. Bên cạnh đó lấy thông tin về các cổng của thiết bị trong bảng IfTable và trả về dưới định dạng JSON
  + Chỉnh sửa một nút:
    - Đường dẫn: /node/edit/
    - Phương thức: POST
    - Tham số: topoId(GET): Id của topology, data(POST): Dữ liệu về nút cần cập nhật lại.
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Lấy deviceId của thiết bị trong data, kiểm tra thiết bị đó trong topology ứng với topoId là do người dùng tự thêm hay được đồng bộ từ proxy. Nếu là do người dùng tự thêm thì cập nhật lại dữ liệu bằng dữ liệu trong data, ngược lại thông báo không thể chỉnh sửa nút này.
  + Xóa một nút:
    - Đường dẫn: /node/delete/
    - Phương thức: POST
    - Tham số: topoId(GET): Id của topology, deviceId(POST): id của nút cần xóa
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Kiểm tra thiết bị với deviceId và topoId là do người dùng tự thêm hay được đồng bộ từ proxy. Nếu là do người dùng tự thêm thì xóa nút đó trong cơ sở dữ liệu, ngược lại thông báo không thể xóa nút.
  + Thêm một nút:
    - Đường dẫn: /node/add/
    - Phương thức: POST
    - Tham số: topoId(GET): Id của topology, data(POST): dữ liệu của node cần thêm vào.
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Đẩy thông tin của nút từ data vào bảng device, lấy deviceId của thiết bị để tiếp tục đẩy thông tin về các cổng của thiết bị vào bảng IfTable. Các bản ghi được đẩy vào có giá trị trường fixed=0 để chỉ có thể thay đổi bằng những tác vụ khác.
* **Cạnh**
  + Chỉnh sửa cạnh:
    - Đường dẫn: /edge/edit/
    - Phương thức: POST
    - Tham số: topoId(GET): Id của topology, data(POST): dữ liệu của edge cần chỉnh sửa.
    - Trả về: Thông báo thành công hay thất bại
    - Cài đặt: Chỉ được phép cập nhật trạng thái hai đầu cổng của hai thiết bị đang liên kết bằng cạnh này. Không giới hạn hai thiết bị đó có cố định hay không.
  + Xóa cạnh:
    - Đường dẫn: /edge/delete/
    - Phương thức: POST
    - Tham số: topoId(GET): Id của topology, connectId(POST): Id của cạnh bị xóa.
    - Trả về: Thông báo thành công hay thất bại
    - Cài đặt: Kiểm tra cạnh với connectId tại tham số trong bảng topology có cố định không, nếu cố định thông báo cạnh đó cố định, nếu không xóa bản ghi khỏi bảng.
  + Thêm cạnh:
    - Đường dẫn: /edge/add/
    - Phương thức: POST
    - Tham số: topoId(GET): Id của topology, data(POST): dữ liệu của node cần thêm vào.
    - Trả về: Thông báo hành động thành công hay thất bại.
    - Cài đặt: Lấy thông tin về hai thiết bị và cổng kết nối hai thiết bị đó và thêm vào bảng topology.

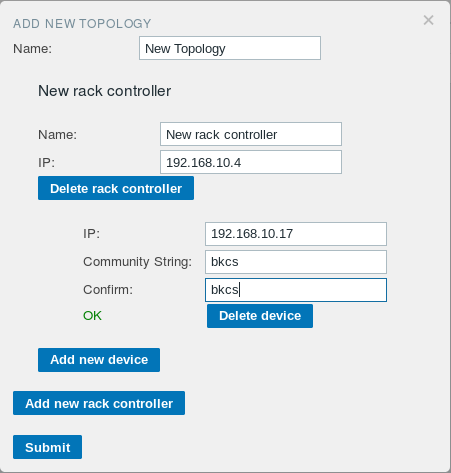
# **CHƯƠNG III: MODULE JS QUẢN LÝ VÀ HIỂN THỊ TOPOLOGY**

1. **Quản lý danh sách topology**
   1. **Các chức năng**

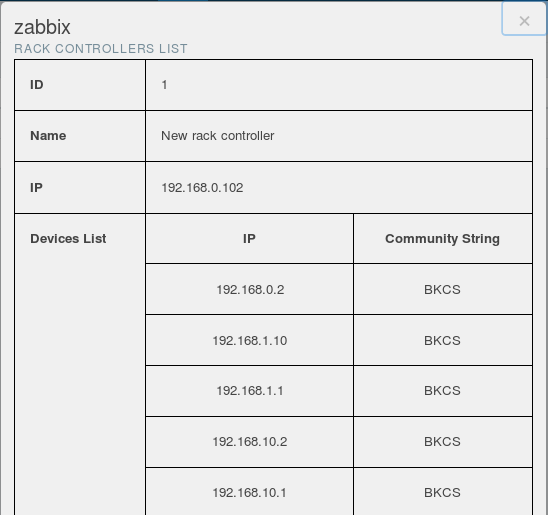
* **Hiển thị danh sách các topology hiện có**
  + Danh sách các topology hiện có được cung cấp bởi web service của server ở đường dẫn /topology/list/. App manager của angularjs có nhiệm vụ hiển thị Id cùng tên của topology đó và cung cấp các chức năng liên quan.



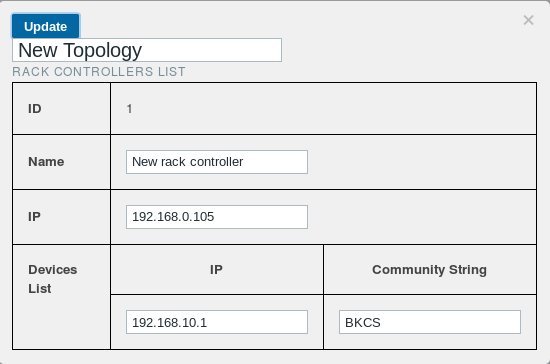
* + Click vào tên một topology sẽ được redirect đến đường dẫn hiển thị topology cộng thêm tham số topoId là ID của topology đó.
  + Các chức năng liên quan tới một topology là xem thông tin, sửa đổi và xóa topology đó
* **Tạo một topology mới**
  + Mở một dialog bằng thư viện ngDialog để cung cấp các tùy chọn cho người dùng tạo một topology mới.



* + Mỗi topology bao gồm các rack controller, mỗi rack quản lý các thiết bị trong rack của mình. Thông tin về rack gồm tên rack và địa chỉ IP của proxy quản lý rack, các thiết bị trong một rack gồm IP và community string.
  + Sau khi điền đầy đủ thông tin và submit, thông tin sẽ được gửi đến web service tại server ở /topology/add để thêm một topology mới vào danh sách.
* **Xem thông tin về một topology**
  + Với một topology trong danh sách, có thể xem thông tin về dach sách các rack và thiết bị trong rack đó:



* + Thông tin về topology được cung cấp bởi web service của server ở đường dẫn /topology/detail với tham số topoId là ID của topology cần xem thông tin.
* **Sửa một topology**
  + Trong quá trình quản lý, có thể IP proxy của một rack controller có thể bị thay đổi, hoặc community string của thiết bị bị thay đổi, khi đó người quản trị có thể cập nhật lại thông tin về topology đó:

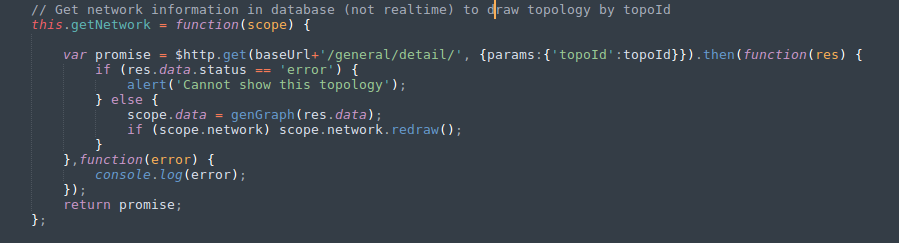


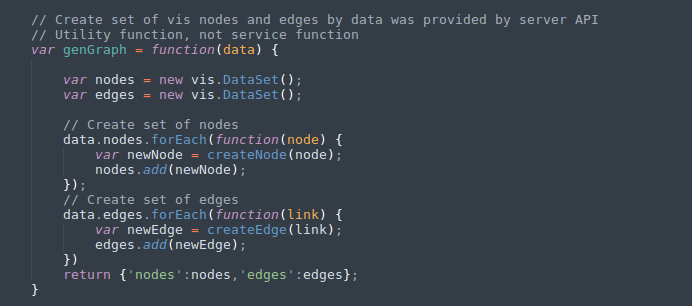
* + Sau khi cập nhật và submit, thông tin sẽ được gửi về server ở đường dẫn /topology/edit/ với tham số là thông tin vừa mới được cập nhật lại.
* **Xóa một topology**
  + Chức năng xóa một topology gửi request tới server ở đường dẫn /topology/delete với tham số là topoId của topology cần xóa.
  1. **Phương thích tích hợp**
* Để tích hợp vào trang web, cần các file sau đây:
  + CSS file: /map/components/dialog/ngDialog.min.css, /map/components/dialog/ngDialog-theme-default.min.css.
  + JS file: /map/components/angular/angular.min.js, /map/components/dialog/ngDialog.min.js, /map/topologyManager.js.
  + HTML:



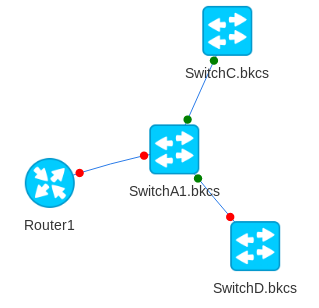
1. **Hiển thị và thao tác với topology**
   1. **Hiển thị topology**

* **Xây dựng tập dữ liệu cho visjs**
  + Để hiển thị được topology cần xây dựng tập dữ liệu của visjs để gọi hàm tạo network của visjs. Tập dữ liệu được lấy từ web service của server ở đường dẫn /general/detail/ với tham số topoId là ID của topology cần hiển thị. Dữ liệu trả về sau đó được cấu trúc lại để phù hợp với visjs.
  + Module ngVis bao gồm một service Data chứa các hàm để giao tiếp với web service của server và directive visNetwork để vẽ topology. Hàm getNetwork trong service Data có nhiệm vụ gọi AJAX request đến API của web service để lấy thông tin về topology, sau đó gọi các hàm hỗ trợ để tái cấu trúc và tạo bộ dữ liệu phù hợp để visjs có thể sử dụng để vẽ network.

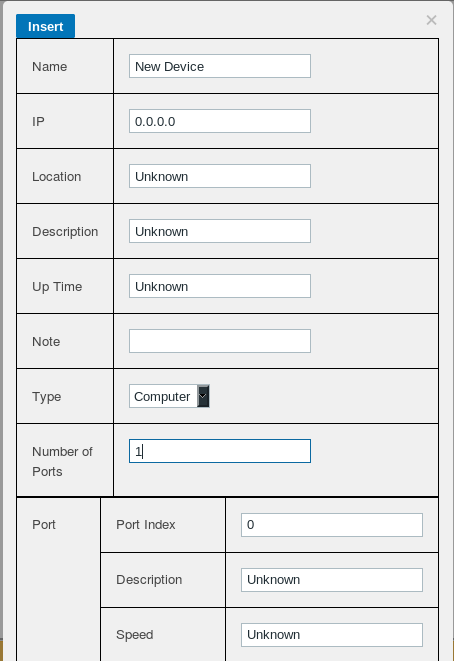




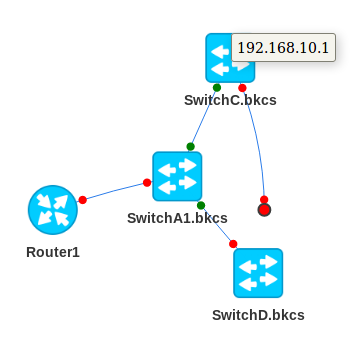
* **Vẽ topology**
  + Module visjs cung cấp hàm để vẽ topology dựa trên tập dữ liệu được cung cấp. Directive visNetwork đóng vai trò là container chứa toàn bộ topology, là nơi hiển thị và thao tác với topology. Trong directive visNetwork sử dụng hàm $watch để bắt bất kỳ thay đổi nào trong tập dữ liệu của topology và ngay lập tức vẽ lại.
  + Icon của các nút được lấy từ /img/, mỗi thiết bị ứng với một icon tùy thuộc vào kiểu của thiết bị, có thể là switch, router, computer hoặc unknown nếu như chưa biết kiểu của thiết bị.
  + Kết nối giữa các nút của visjs không có chức năng hiển thị trạng thái up/down của cổng kết nối, do đó cần viết thêm vào mã nguồn của visjs để thêm chức năng thay đổi màu của 2 đầu liên kết dựa trên trạng thái cổng của thiết bị trong liên kết.
  + Tên của thiết bị được hiển thị bên dưới icon của nút, khi rê chuột vào nút sẽ hiển thị IP của thiết bị, click vào nút hoặc cạnh để chọn, phục vụ các chức năng thao tác với nút và cạnh. Khoảng cách giữa các nút và phương thức hiển thị có thể tùy chỉnh trong biến option của app TopologyViewer.



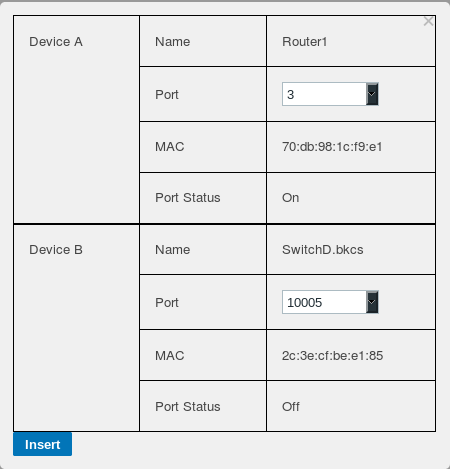
* + Visjs cung cấp đầy đủ các chức năng thu, phóng network, kéo thả các nút để tùy chỉnh hình dạng network và tự động điều chỉnh khoảng cách giữa các nút để tránh các nút chồng chất lên nhau.
  + Kích thước của container chứa topology có thể tùy chỉnh sử dụng CSS đối với directive visNetwork trong file /style/network.css.
  + Khi trang web được tải, app topologyViewer sẽ tự động gọi hàm getNetwork trong service Data để vẽ topology lần đầu tiên, sau đó với mỗi thao tác làm thay đổi dữ liệu trong database ở server, hàm getNetwork sẽ được gọi lại để vẽ lại network.
  1. **Các thao tác với topology**
     1. **Thao tác chung với topology**
* **Thêm nút**
  + Thao tác thêm nút được thực hiện bằng thao tác kéo thả với sự hỗ trợ của module ngDraggable của angularjs. Các kiểu nút có biểu tượng tương ứng ở thanh toolbox bên trái khung topology. Khi kéo một biều tượng thả vào khung topology, một dialog sẽ được mở lên để nhập các thông số cho nút mới như tên, IP… và số lượng port của thiết bị cũng như thông tin chi tiết của các port đó.



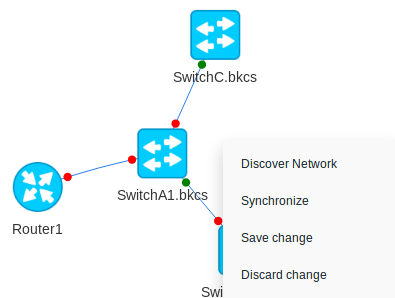
* + Sau khi nhập đầy đủ thông tin và nhấn nút Insert, controller của ngDialog sẽ gọi hàm addNode của service Data để gọi API /node/add/ của server để thêm một nút vào cơ sở dữ liệu đồng thời gọi hàm getNetwork để vẽ lại topology với nút mới vừa được thêm vào.
* **Thêm cạnh**
  + Tương tự thêm nút, thao tác thêm cạnh cũng được xây dựng với sự hỗ trợ của module ngDraggable, tuy nhiên để dễ dàng thao tác cho người sử dụng, thao tác này được cài đặt chồng lên thao tác thêm cạnh chuẩn của visjs. Khi kéo biểu tượng cạnh vào khung topology, tất cả các nút của topology sẽ được chọn (in đậm), khi đó tiếp tục kéo một đường từ một nút đến một nút khác để tạo cạnh mới giữa 2 nút.



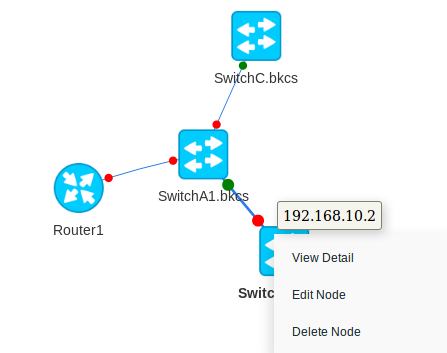
* + Khi kết nối thành công 2 nút, một dialog được mở lên để chọn cổng mà 2 thiết bị sẽ kết nối với nhau trong danh sách các cổng của thiết bị. Thông tin về cổng của thiết bị được lưu trong cache nếu thiết bị đã từng được lấy thông tin, ngược lại hàm getNodeDetail của service Data sẽ được gọi để lấy thông tin về thiết bị và lưu vào cache. Với mỗi cổng được chọn sẽ có các thông tin về tên thiết bị, địa chỉ MAC, số hiệu cổng, trạng thái up/down của cổng để người sử dụng dễ dàng nhận ra cổng cần tìm. Bắt buộc phải chọn cổng để có thể tạo một cạnh mới.
  + Sau khi chọn được cổng và nhấn Insert, hàm addEdge của service Data sẽ được gọi để gọi đến API /edge/add/ của webservice để thêm liên kết mới vào cơ sở dữ liệu đồng thời gọi hàm getNetwork để vẽ lại topology với cạnh mới vừa được thêm.



* + Lưu ý: Thêm nút và thêm cạnh là 2 thao tác duy nhất với topology mà không sử dụng menu chuột phải, tất cả các thao tác tiếp theo đều được gọi bằng cách chọn trong menu dropdown khi click chuột phải. Menu chuột phải được xây dựng tùy theo đối tượng nào đang được chọn. Nếu không nút nào hoặc không cạnh nào được chọn, các chức năng khi click chuột phải sẽ là các chức năng chung đối với topology (menu chung), nếu có một nút được chọn thì bao gồm các chức năng đối với nút được chọn (menu nút), nếu một cạnh được chọn thì các chức năng là của cạnh được chọn (menu cạnh).
  + Menu chuột phải được xây dựng tại factory contextMenu của app topologyViewer, gồm các hàm để thực hiện các chức năng của 3 loại menu. Các hàm chủ yếu có chức năng xử lý dữ liệu để gửi đi các API của server và xử lý dữ liệu nhận lại để hiển thị ra người dùng. Dữ liệu được hiển thị bằng module ngDialog của angularjs với template và controller riêng. Bên cạnh đó đây còn là nơi cache lại các dữ liệu có thể dùng nhiều lần để tăng tốc độ xử lý.
* **Discover mạng**
  + Dữ liệu để vẽ topology đều là dữ liệu trong database ở server được tổng hợp từ các rack controller. Server không tự động lấy dữ liệu định kỳ ở rack mà chỉ lấy khi có lệnh từ người sử dụng qua chức năng discover mạng. Đây là chức năng ở menu chung, hàm discover của service Data sẽ được gọi để gọi đến API /general/discover/ của web server, server sẽ gửi tín hiệu đến proxy của các rack controller để proxy dùng snmp lấy thông tin về các thiết bị trong rack. Tuy nhiên proxy của các rack controller sau khi lấy dữ liệu sẽ chỉ lưu trong cơ sở dữ liệu của mình chứ không gửi lại lên server, dữ liệu tại database trên server vẫn là dữ liệu cũ.
* **Đồng bộ dữ liệu giữa server và các proxy**
  + Như đã nói ở trên, chức năng discover mạng chỉ gửi tín hiệu để proxy tại các rack controller lấy thông tin về các thiết bị trong rack của mình chứ không thực hiện đồng bộ dữ liệu từ proxy về server, do đó cần thêm chức năng đồng bộ giữa server và các proxy. Chức năng synchronize trong menu chung sẽ gọi đến hàm sync của service Data để gọi API /general/synchronize/ để đồng bộ dữ liệu từ proxy về rack. Sau khi đồng bộ thành công topology sẽ được vẽ lại.
* **Lưu/hủy bỏ thay đổi** 
  + Các thao tác với tập dữ liệu trong database của server đều là thao tác tạm, có thể được roll back. Chức năng save change trong menu chung để gọi API /general/save/ để commit các thay đổi vừa thực hiện trong database, khi đó các thay đổi sẽ không thể roll back được nữa. Tương tự, chức năng discard change trong menu chung để hủy các thay đổi vừa thực hiện. Lưu ý sau khi thực hiện thay đổi với topology thành công cần gọi chức năng save change ngay để tránh mất mát dữ liệu.

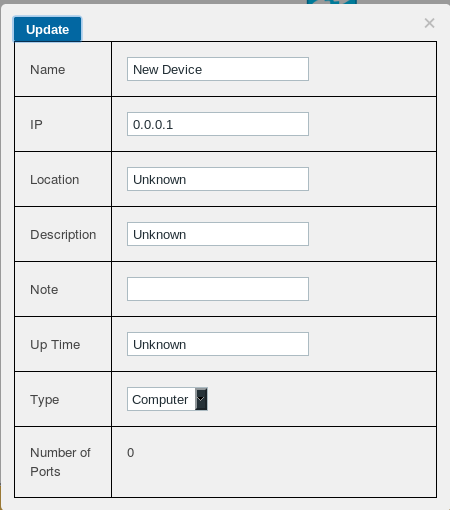


* + 1. **Thao tác với nút**
* **Xem thông tin**
  + Thông tin về nút để vẽ topology chỉ là thông tin cơ bản về tên, kiểu và IP của thiết bị, để xem thông tin chi tiết và các cổng cần click vào thiết bị đó để chọn nút sau đó click chuột phải, chọn chức năng View Detail. Hàm getNodeDetail của service Data sẽ được gọi để gọi API /node/detail/ của webservice của server lấy thông tin về thiết bị đó lưu vào cache. Lần tiếp theo cần sử dụng thông tin chi tiết của nút sẽ được lấy từ cache để tăng tốc độ xử lý, giảm thiểu số lượng request gửi đến server. Dữ liệu sẽ được hiển thị bằng module ngDialog của angularjs.

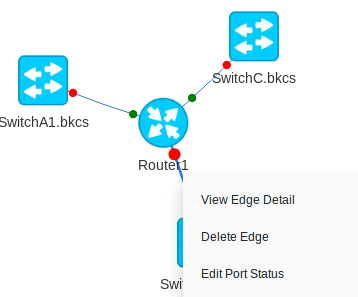




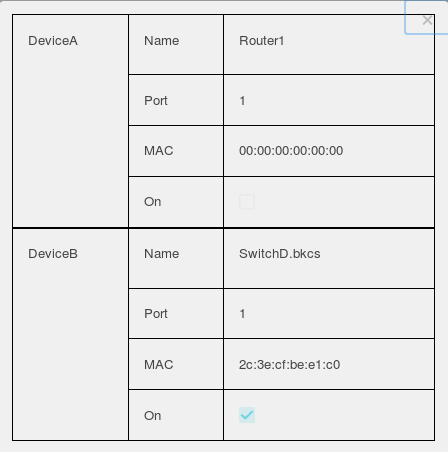
* **Sửa thông tin nút**
  + Các nút và cạnh trong topology được chia làm 2 loại: cố định và tùy chỉnh. Loại cố định là các nút và cạnh được lấy thông tin ở các rack controller sau đó tổng hợp về server và không được phép thay đổi thông tin, loại tùy chỉnh là loại được thêm vào bằng các chức năng thêm nút và thêm cạnh và có thể được thay đổi thông tin. Thông tin về thiết bị lấy được ở server có bao gồm kiểu cố định hay tùy chỉnh của thiết bị. Khi chọn chức năng Edit Node ở menu nút, nếu nút đó là thiết bị cố định sẽ thông báo lại và không cho phép sửa. Nếu nút đó là nút thiết bị tùy chỉnh, một dialog sẽ được mở lên cho phép người sử dụng thay đổi thông tin về thiết bị. Sau khi hoàn tất thay đổi và chọn Update, thông tin mới sẽ được gửi đến hàm editNode của service Data để gửi đến server và cập nhật trong database.



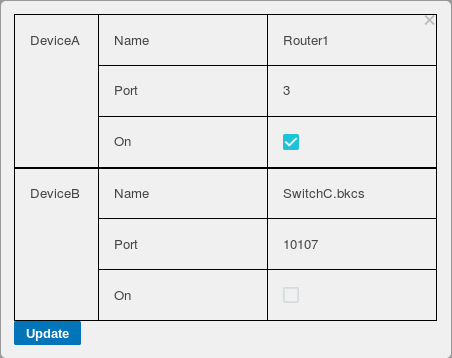
* **Xóa nút**
  + Chức năng xóa nút cũng chỉ áp dụng với những nút tùy chỉnh, trong trường hợp người sử dụng thêm vào nhưng không cần sử dụng tới nữa. Khi gọi tới chức năng Delete Node của menu nút, hàm deleteNode của service Data sẽ được gọi để gửi thông tin nút cần xóa đến server và xóa nút đó khỏi cơ sở dữ liệu.
    1. **Thao tác với cạnh**
  + Các thao tác chính với cạnh là xem thông tin cạnh, xóa cạnh và sửa trạng thái cổng 2 đầu thiết bị của cạnh:



* **Xem thông tin cạnh** 
  + Thông tin về cạnh bao gồm tên của 2 thiết bị được kết nối bởi cạnh đó, số hiệu cổng và địa chỉ MAC của 2 cổng kết nối 2 thiết bị cùng trạng thái up/down hiện tại của cổng đó.
  + Thông tin về cạnh chỉ lấy từ thông tin cơ bản để vẽ topology, ko cần lấy thêm ở server. Hàm getEdgeDetail của service Data có nhiệm vụ tổng hợp lại từ tập dữ liệu của topology và trả về những thông tin cần thiết để hiển thị. Tuy nhiên thông tin đã được tổng hợp vẫn được lưu vào cache để tăng tốc độ xử lý cho những lần truy nhập sau.



* **Xóa cạnh**
  + Cũng giống xóa nút, cạnh cũng chỉ có thế xóa nếu đó là cạnh được người sử dụng tự thêm vào, còn các cạnh cố định được tổng hợp từ các rack controller thì không thể bị xóa.
* **Sửa trạng thái cổng**
  + Những thông tin khác của cạnh không thể thay đổi, nếu thay đổi thì tương đương với tạo cạnh mới, chỉ có trạng thái 2 đầu cổng của 2 thiết bị có thể được thay đổi do trong quá trình sử dụng có thể một cổng nào đó bị tắt hoặc được bật. Chức năng này có thể hoạt động cả với những cạnh cố định và người sử dụng phải hoàn toàn chịu trách nhiệm về thay đổi của mình.



* + Thông tin về trạng thái mới của cổng sẽ được hàm editEdge của service Data gửi đến API /edge/edit/ của server để cập nhật lại trong database. Topology sẽ được vẽ lại để thay đổi màu cổng đó nếu tác vụ thành công.

# **CHƯƠNG IV: KẾT QUẢ ĐẠT ĐƯỢC VÀ PHƯƠNG HƯỚNG PHÁT TRIỂN**

1. **Kết quả đạt được**
   1. **Web service tại server**
   * Xây dựng thành công web service với đầy đủ các chức năng cần thiết để xây dựng và quản lý danh sách các topology cũng như các chức năng để giao tiếp với proxy quản lý các rack controller để lấy thông tin về các thiết bị trong mạng đồng thời tổng hợp và cung cấp các API để module javascript có thể lấy thông tin phục vụ vẽ và thao tác với topology mạng.
   1. **Module JS quản lý và hiển thị topology**
   * Xây dựng thành công module javascript với giao diện thân thiện, trực quan, dễ sử dụng, cung cấp cho người dùng môi trường làm việc hiệu quả để quản lý hệ thống mạng qua topology của mạng.
2. **Phương hướng phát triển**
   * Thêm chức năng SSH tới một thiết bị qua giao diện web để cấu hình trực tiếp thiết bị đó.