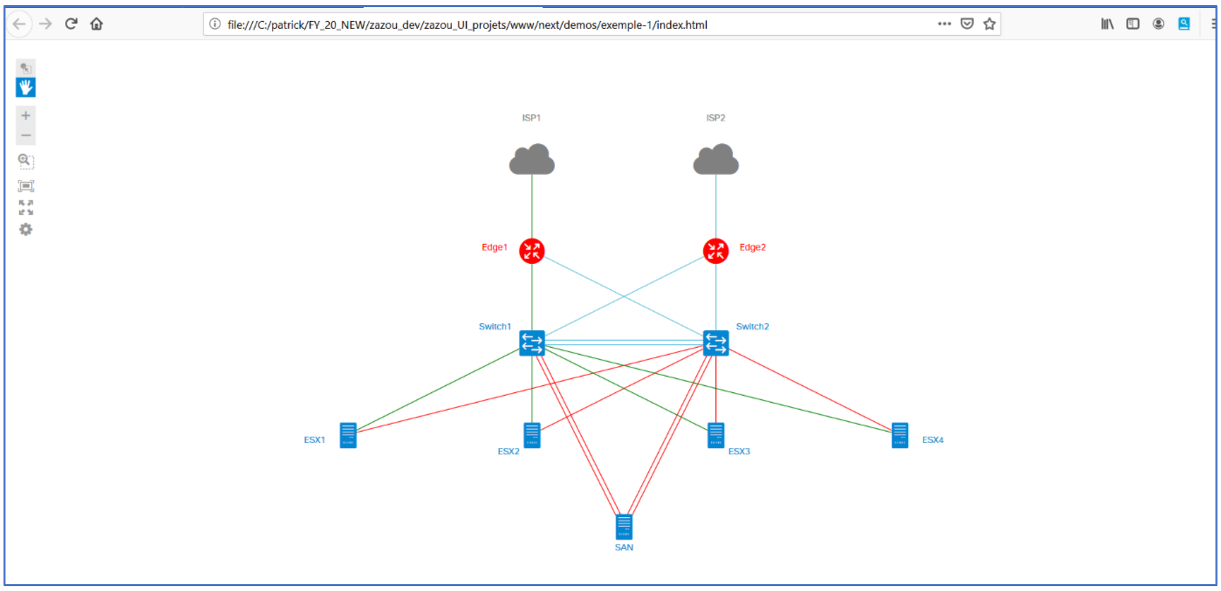
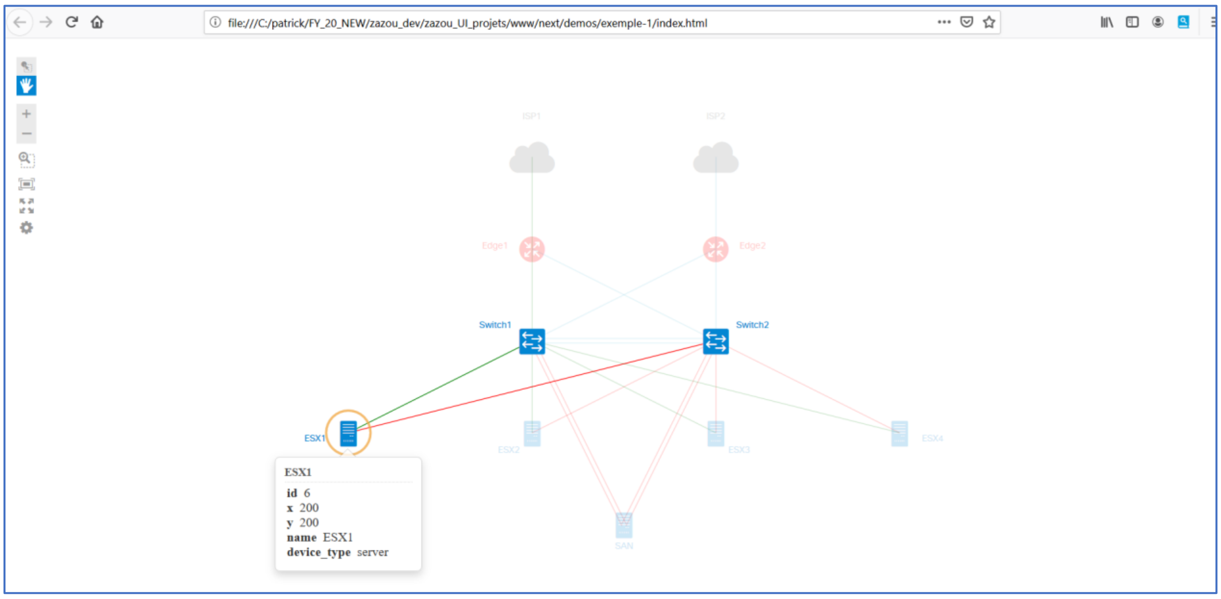
Javascript example in 3 minutes

One of the good things in Javascript is that you don't need a web server to be able to read a javascript document.

That means that you can create a document which will not be a PDF neither a Word document neither a html document, but you will be able to display a nice a dynamic contents like graphs, without needing anything else than you browser.

The example bellow shows you that. This document is fully dynamic, you can click on icons


file:///C:/patrick/FY_20_NEW/zazou_dev/zazou_UI_projets/www/next/demos/exemple-1/index.html


file:///C:/patrick/FY_20_NEW/zazou_dev/zazou_UI_projets/www/next/demos/exemple-1/index.html

ESX1

id 6
x 200
y 200
name ESX1
device_type server

Really cool if you only need to show graphs from of pre calculated results

BUT ! if you need to input data, and compute them, dynamically generate new graphs, then at some point you will need to setup a full functional web server to support your javascprit application and probably a SQL database behind

Let’s keep this example, it is very good to introduce how to create your javascript application.

And after that, we will add a web server behind.

How to Create a basic Javascript Application

This just assembly

You need :

A terminal which is Your Browser

An html file which will be opened by your browser : **index.html**

This html files will

- display the javascript generated content
- load needed javascript libraries

A CSS file which will format how the html content which will be displayed : **style.css**

The javascript application file : **app.js**

A topology data file : **topology.js**

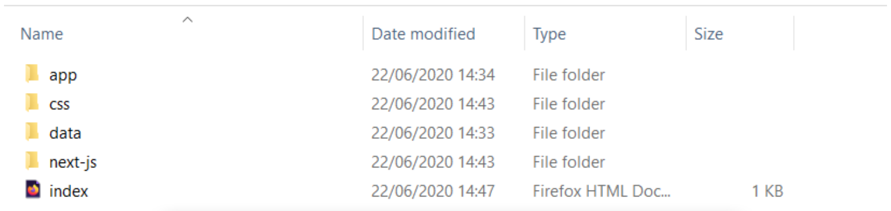
Javascript Librairies files **(Cisco Next UI)**

- Which load into your browser all functions needed to compute, and display all data and objects, used and output by your application
- In our example we will use **Cisco Next UI**

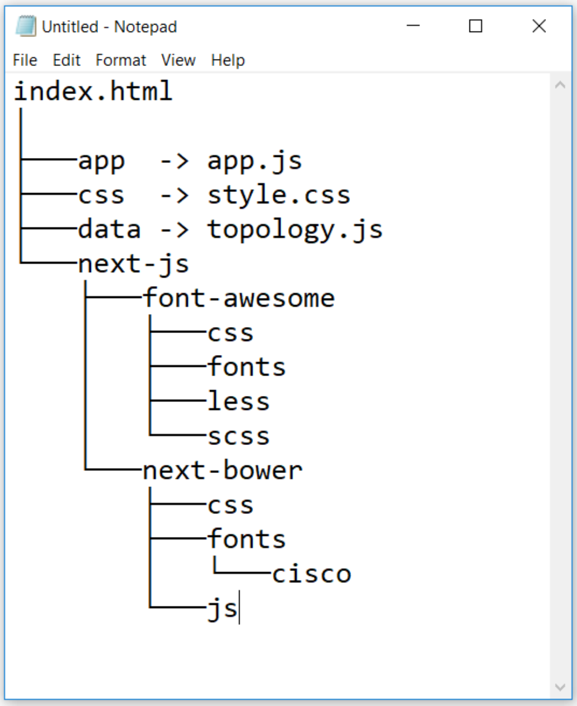
The application structure

Your application is basically an assembly of all of above mentioned files. The entry point of this application is the **index.html** files. You will open this file with your browser in order to run your application.

And here under here is the directory structure we are going to build



Name	Date modified	Type	Size
app	22/06/2020 14:34	File folder	
css	22/06/2020 14:43	File folder	
data	22/06/2020 14:33	File folder	
next-js	22/06/2020 14:43	File folder	
index	22/06/2020 14:47	Firefox HTML Doc...	1 KB



```
index.html
│
├────app   -> app.js
├────css   -> style.css
├────data  -> topology.js
└────next-js
        ├────font-awesome
        │       ├────css
        │       ├────fonts
        │       ├────less
        │       └────scss
        └────next-bower
                ├────css
                ├────fonts
                │       └────cisco
                └────js
```

Index.html

Create the root directory for your application

Create a file named **index.html** and copy and paste into it the following content

```
<!DOCTYPE html>
  <head>
                <link rel="stylesheet" href="./next-js/next-bower/css/next.css">
                <link rel="stylesheet" href="./css/style.css">
                <script type="text/javascript" src="./next-js/next-bower/js/next.js"></script>

  </head>
  <body>
    <div id="topology-container"></div>
    <script src="./data/topology.js"></script>
    <script src="./app/app.js"></script>
  </body>
</html>
```

Remark : look into this index.html file path from where we ask the browser to load all application files

That means that you can create any directory structure as you want as long as you mention the correct path to load needed file into your index.html file

App.js

This file is the application. This is a Next UI application, full javascript.

Create a directory named **app**

And create into it a file names **app.js** with the following content.

I won't comment here the details of this file. BUT I will do in another tutorial.

```
(function(nx){

        // intialisation : instantiate next app
        const app = new nx.ui.Application();

        // configuration objects we define here under the details of the NeXt View
        const topologyConfig = {
          // configuration of canvas size
           width: window.innerWidth,
           height: window.innerHeight,
           // configuration for nodes
           nodeConfig: {
                  label: "model.name",
                  iconType: "model.device_type",
                  color: "model.color"
        },
           // configuration for links
          linkConfig: {
                  linkType: "straight",
                  color: "model.color"
        },
        // if true, the nodes' icons are shown, a dot is shown instead
        showIcon: true,
        };

        // instantiate Topology class
        const topology = new nx.graphic.Topology(topologyConfig);

        // load topology network data from ./app/topology.js
        topology.data(topologyData);

        // bind the topology object to the app
        topology.attach(app);

        // app must run inside a specific container. In our case this is the <div> with id="topology-container" in
the index.html file
        app.container(document.getElementById("topology-container"));

})(nx);
```

CSS Cascading Style Sheet

Every Web contain use CSS files to make them nice and beautiful.

It’s the same here, even this example doesn’t really need sophisticated CSS, let’s create one.

in other Next-UI projects we will define into it much more complex styles.

Create a sub directory named **css** .

Create into it a file named **style.css** and copy and paste the following contain into it.

```
body,
html {
  height: 100%;
}
```

Topology.js

Create a subdirectory name

This file content all nodes and link details.

Create a file named **topology.js** and copy and paste the following contain into it.

I won’t comment the details of this files, I will do it in another tutorial. But this one is very easy to understand. Have a look to it.

It contains just nodes and links definitions.

Is this example Every node is defined by an id, x position, y position, a name, a type, a color.

It’s the same for links, links connect nodes, they are defined by a source node id, a target node id and a color.

```
const topologyData = {
   nodes: [
      // ISPs
      { id: 0, x: 400, y: -100, name: "ISP1", device_type: "cloud", color: "grey" },
      { id: 1, x: 600, y: -100, name: "ISP2", device_type: "cloud", color: "grey" },
      // Routers
      { id: 2, x: 400, y: 0, name: "Edge1", device_type: "router", color: "red" },
      { id: 3, x: 600, y: 0, name: "Edge2", device_type: "router", color: "red" },
      // Switches
      { id: 4, x: 400, y: 100, name: "Switch1", device_type: "switch" },

      { id: 5, x: 600, y: 100, name: "Switch2", device_type: "switch" },

      // Servers

      { id: 6, x: 200, y: 200, name: "ESX1", device_type: "server" },

      { id: 7, x: 400, y: 200, name: "ESX2", device_type: "server" },

      { id: 8, x: 600, y: 200, name: "ESX3", device_type: "server" },

      { id: 9, x: 800, y: 200, name: "ESX4", device_type: "server" },

      // SAN

      { id: 10, x: 500, y: 300, name: "SAN", device_type: "server" }

   ],

   links: [

     // WAN to routers

     { source: 0, target: 2, color: "green" },

     { source: 1, target: 3 },

     // Routers to switches

     { source: 2, target: 4, color: "green" },

     { source: 2, target: 5 },

     { source: 3, target: 4 },

     { source: 3, target: 5 },

     // Switches to Switches

     { source: 4, target: 5 },

     { source: 4, target: 5 },

     // Servers to Switches

     { source: 6, target: 4, color: "green" },

     { source: 6, target: 5, color: "red" },

     { source: 7, target: 4, color: "green" },

     { source: 7, target: 5, color: "red" },

     { source: 8, target: 4, color: "green" },

     { source: 8, target: 5, color: "red" },

     { source: 9, target: 4, color: "green" },

     { source: 9, target: 5, color: "red" },
     // SAN to Switches

     { source: 10, target: 4, color: "red" },

     { source: 10, target: 4, color: "red" },

     { source: 10, target: 5, color: "red" },

     { source: 10, target: 5, color: "red" }

   ]

 };
```

You have probably understood that the only file to modify, if you want to display any other network diagram is this file ! and for those of you who are already familiar with javascript you probably understood that you can change on the fly the value of the **topologyData** variable in order to dynamically display a new network.

NeXt-UI libraries

The **app.js** file is the application file, it calls functions which reside into Next-UI libraries. And this the way that is always work in javascript.

All javascript functions must have been loaded by the browser before they can be called.

Look at the **index.html** file in the <head> section. This is where you can see instructions which loads the needed Next-UI librairies.

You can see that we load one Next-UI CSS file and One Next-UI javascript file.

Remark. The <head> part in the html file is not the only section from where we can load the javascript file. Needed javascript files can be loaded from everywhere in the html file. We just need to load them before using their functions.

```
  <head>
                <link rel="stylesheet" href="./next-js/next-bower/css/next.css">
                <link rel="stylesheet" href="./css/style.css">
                <script type="text/javascript" src="./next-js/next-bower/js/next.js"></script>

  </head>
```

So you probably understand that you can load the needed javascript resources from either your local disk, or from an URL on the INTERNET.

Go thru the following operations :

As the goal of this tutorial is Javascript and Not Next-UI. Let's do some voluntary uncommon operations.

let's install the needed NEXT-UI libraries into our local disk.

STEP 1 – download Next-UI libraries and resources

Let’s do this :

Go to the following URL :

https://github.com/NeXt-UI/next-tutorials

(By the way, you will find here a nice NeXt-UI tutorial !)

Git Clone the content :

And unzip the **next-tutorials-master.zip** file some where into you hardrive.

Look at the **index.html** from which location we will load the NeXt-UI libraries.

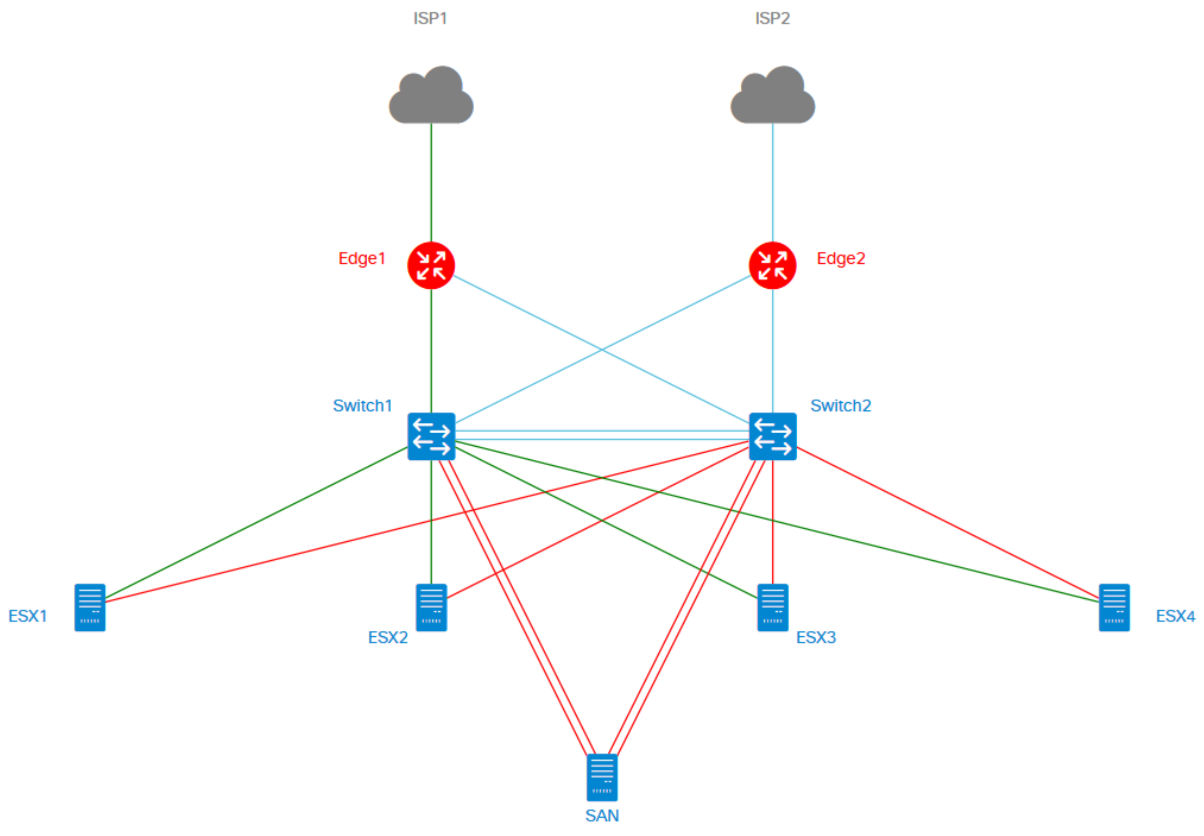
So, this is very straight forward, find into the unzip resources, the subfolder named **bower_component** (next-tutorials-master\demos\common\bower_components)

Copy the 2 folders contained in this folder

- font-awesome
- next-bower

And we are ready to test .

Open the **index.html** with any browser and you should see the network appearing :



Remark : The NeXt-UI library we used is not the last version.

As an exercise, your challenge is now to use the last version of the Next-UI library.

You can download it here under :

https://d1nmyq4gcgsfi5.cloudfront.net/fileMedia/025dc509-8f2a-474a-b6d8-75e73ecbd6ac/NeXt_trial.zip

Create a new subfolder named **new_next_js**

And copy into it all needed resources

Hints : you need the javascript file, the css as well and don't forget the fonts

Delete (or rename) the **next-js** subfolder.

And test ! you should see again the network

What Next ?

From here change the network topology. Add some new icons

For you reference have a look to the following URL and change some icons

https://d1nmyq4gcgsfi5.cloudfront.net/site/neXt/discover/demo/