LAB 2.

Part 1. Installing and configuring remote OS on VBox.

1. After installation of virtual machine, start OS, and install openSSH server. In Ubuntu terminal run next command:

```
sudo apt install openssh-server
```

Verify that ssh service running:

```
sudo systemctl status ssh
```

If service is not started, you can start (stop, restart) the service manually via command:

```
sudo systemctl ssh start
```

For allowing incoming SSH connections the Uncomplicated Firewall (next UFW) should be configured. Try to check the status of UFW with use of following command:

```
sudo ufw status
```

Then enable UFW firewall by typing:

```
sudo ufw enable
```

And run next command to allow ssh connections:

```
sudo ufw allow ssh
```

Also need to allow some common ports 22 and 80:

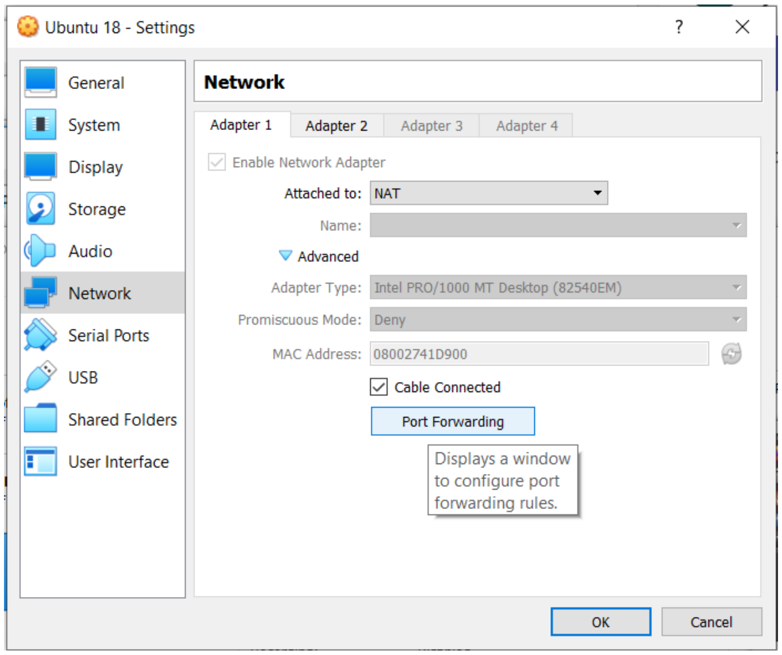
```
sudo ufw allow 22
sudo ufw allow 80
```

Now we are ready to configure SSH itself. Power off the machine and follow next instructions.

2. The SSH connection to a VM can be by configuring Networking of VM. There are a lot of options for it, but we will use the following:

*1) **via setup NAT port forwarding**.*

By default, our VM should have already at least one interface which is using NAT. You can check it by going to the settings of VM, select Network and first adapter expected to be attached to **NAT**. In order to setup port forwarding for it, expand Advanced menu and click on the **Port Forwarding** button.



Attached to: NAT
Name:
Advanced
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)
Promiscuous Mode: Deny
MAC Address: 08002741D900
Cable Connected
Displays a window to configure port forwarding rules.

Add a new Rule to the table, with next values:

Name: SSH

Protocol: TCP

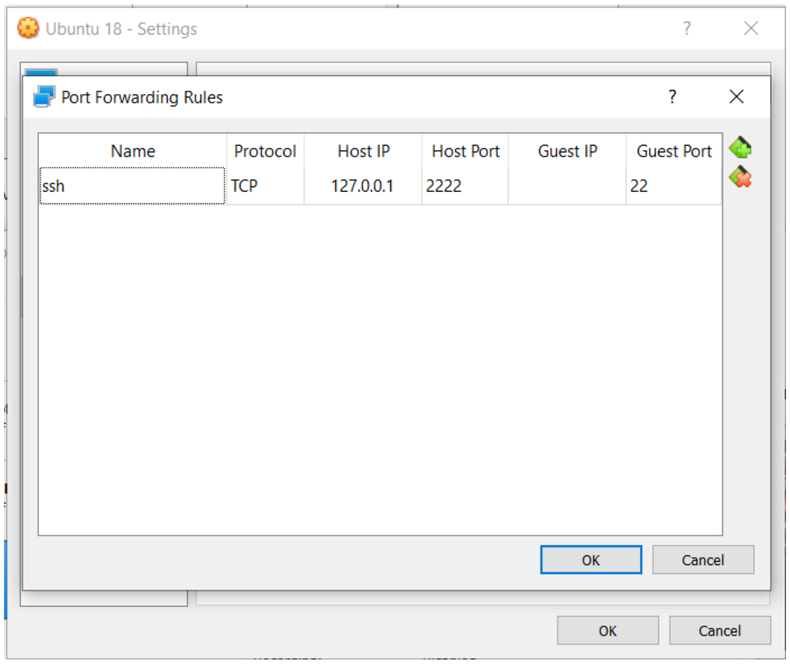
Host IP: 127.0.0.1 (*)

Host Port: 2222

IP Guest: Empty

Port Guest: 22

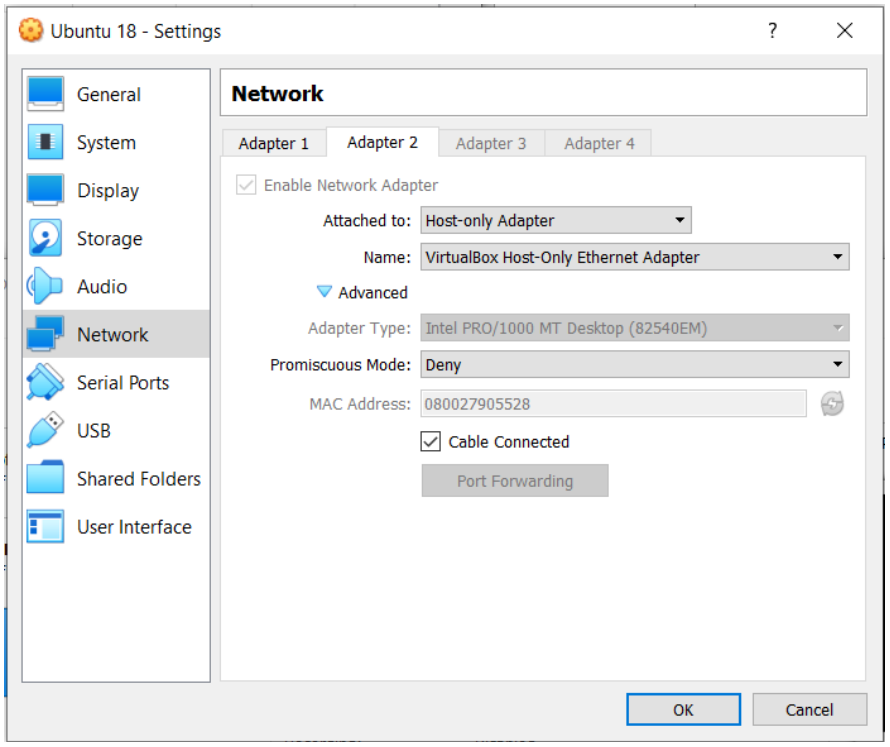
* Usually it will be provided value, but to be sure you can check it in Ubuntu's terminal by running `sudo ip a`.



Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
ssh	TCP	127.0.0.1	2222		22

2) ***via Host Only with using VM's IP address.***

Back to Settings > Network and either enable Adapter 2 or change Adapter 1 and make it attached to **Host Only Adapter**. Save changes.


Ubuntu 18 - Settings

Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

☑ Enable Network Adapter

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

Advanced

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Promiscuous Mode: Deny

MAC Address: 080027905528

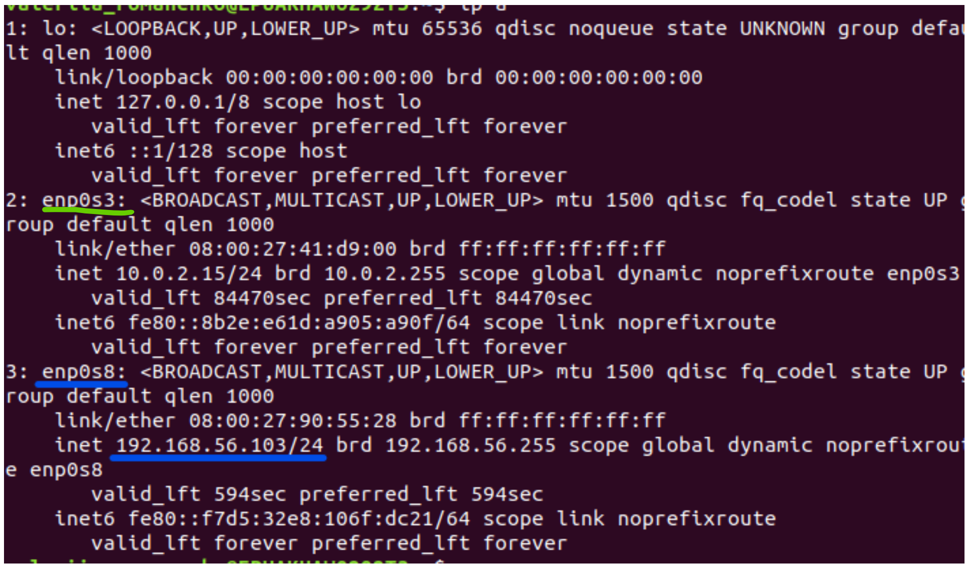
☑ Cable Connected

Port Forwarding

Now start Ubuntu, open terminal and get the VM's IP addresses:

sudo ip a

If you setup machine's networks correctly it will show three adapters — lo (loopback network interface) is to connect to itself, enp0s3 (NAT) is for connecting to Internet and enp0s8 (Host-Only) is for connecting to host and other VMs. So, we will need **enp0s3** and **enp0s8** IP's:


```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defa
lt qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
roup default qlen 1000
    link/ether 08:00:27:41:d9:00 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 84470sec preferred_lft 84470sec
    inet6 fe80::8b2e:e61d:a905:a90f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
roup default qlen 1000
    link/ether 08:00:27:90:55:28 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.103/24 brd 192.168.56.255 scope global dynamic noprefixrou
e enp0s8
       valid_lft 594sec preferred_lft 594sec
    inet6 fe80::f7d5:32e8:106f:dc21/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Finally, connect to remote OS. For connecting via NAT adapter:

```
ssh -p 2222 [user]@127.0.0.1
```

And for connecting using Host-Only adapter:

```
ssh [user]@[enp0s8_ip]
```

3. You can ping the VM's server by IP from host OS to make sure it's accessible from outside.
4. [**optional**] configure static IP address for your server.

Part 2. Configure SSH.

1. Create a new user with **root** privileges: **username**: *sshuser*, **password**: *sshpassword* . SSH to VM's OS and:
   a. Become the *root user*:
      ```
      sudo su
      ```
   b. Create new user:
      ```
      sudo adduser sshuser
      ```
   c. Add created user to *sudo* group:
      ```
      usermod -aG sudo sshuser
      ```
   d. Modify **/etc/ssh/sshd_config** allowing ssh-connection only for sshuser:
      ```
      sudo vim /etc/ssh/sshd_config
      ```
      Add a new line:
      ```
      AllowUsers sshuser
      ```
      Disable Password Authentication, by uncommenting line and changing *yes* to *no*:
      ```
      PasswordAuthentication no
      ```
   e. Save file and exit. Then restart SSH service:
      ```
      sudo systemctl restart ssh
      ```
   f. Terminate current SSH connection. Make sure that your user created during OS installation is now **not able to connect via SSH**, and that **sshuser – able** to do that. You should see in first case *Permission denied (publickey)* error, and in second – successful started SSH session.

2. Setup SSH connection using SSH key, which providing a secure way of logging to remote servers:
   a. On a local machine generate a new ssh-key:
      ```
      ssh-keygen
      ```
   b. Add a public key for sshuser, it will require to enter user's password:
      ```
      ssh-copy-id -i [path_to_generated_key.pub] sshuser@[remote_host_ip]
      ```
   c. Connect to remote OS via SSH using this public key. The system should not ask for a password as it is negotiating a secure connection using the SSH keys:

```
ssh sshuser@[remote_host_ip]
```

d. Modify **~/.ssh/config** on your PC and create an alias for SSH connection. Add:

```
Host ubuntu-sshuser
   HostName [remote_os_ip]
   User sshuser
   IdentityFile [absolute_pass_to_generated_ssh_key_file]
```

e. Now you can SSH into remote OS using:

```
ssh ubuntu-sshuser
```

Part 3. Preparing Back-End & Front-End apps.

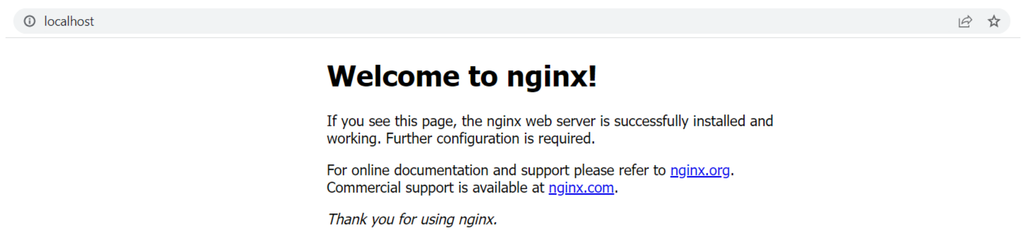
1. For further work you will need 2 apps: Front-End app and Back-End for it. You can use your personal/own REST API server and related single page app (SPA) if you want. Or there is possible to use already existing ones, from the repositories of EPAM JS Competency Center.
2. Set up Back-end app:
   a. Make sure you are on or checkout to branch ***feat/products-api****.*
   b. Store and use the following *environmental variables*:
      i. **APP_PORT** – define an application port on which your server will keep connection and handle requests. E.g., 8080.
      ii. **NODE_ENV** – identifies current application's environment.
      iii. **ENV_CONFIGURATION** – identifies a name of current configuration that application server must use (if any).
   c. Add a necessary npm script(s) and to and start your server. In case using existing app from Competency Center set up project to use described variables.
   d. Use either nodemon or pm2 **to start/re-start** your app (and track when source code is changed).
   e. Start server app.
3. Setup client app (choose shop app with framework you'd prefer more)**:**
   a. **Connect** client app with API, by add a necessary actual URLs in client app (for Angular modify environments files) to make FE app sends requests to NodeJS REST API, which you have just configured.
   b. Check that data requested and displayed successfully and all from available CRUD operations working well, by running and testing both apps.

Part 4. Nginx setup.

1. Download, install (Windows installation) and configure Nginx on your workstation or laptop.
2. For manipulating with nginx commands on Windows enter the folder extracted from downloaded zip archive, star here terminal and run:

```
start nginx
```

Also, might be helpful, add path to this folder to PATH environment variables. After starting nginx open your browser and go to localhost to check whether the installation has been successful. If you see the same as like on the screenshot below - its mean nginx working, and there are no problems with installation.


localhost

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

3. Implement Nginx configuration which will serve FE application and necessary assets:
    a. Go to the root nginx folder and open ***nginx.conf*** file in the ***conf*** folder.
    b. In that file located already prepared but partially commented out templates for servers. Default configuration is responsive for displaying nginx page on localhost. For adding custom configurations for our BE and FE apps may be used one of the following approaches:
        i. Uncommenting existing/adding new configuration in this default config file.
        ii. Create separate file with custom configurations and include them into default config, by adding to *https* section next line for each added file:

        ```
        include "%path_to_project_folder%/%path_to_your_conf_file%/*.conf"
        ```

    Pay attention, that if including, content of the separate config file should not anymore have http section. Start from adding server's one.

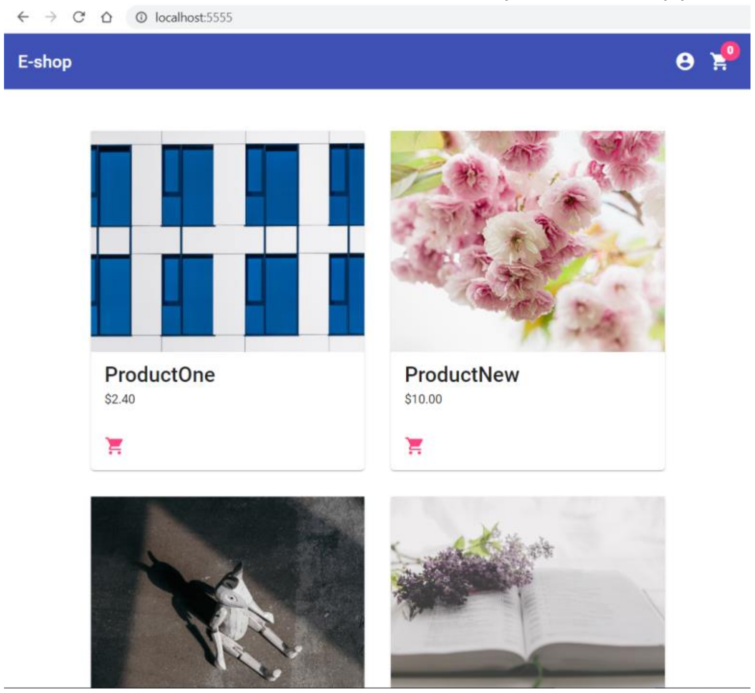
    c. Setup nginx to serve your running front-end application. In this guide we will use separate file for nginx server's configuration, stored in project's directory. Add next configuration to your .config file for local development:

```
upstream front_end_app {
   server localhost:4200;
 }

 server {
   listen 5555;

   location / {
         proxy_pass http://front_end_app;
         proxy_set_header Upgrade $http_upgrade;
         proxy_set_header Connection $http_connection;
         proxy_set_header Host $host;
   }
 }
```

After each change to nginx config file, need to reload nginx by command:

```
nginx -s reload
```

Now go to *localhost:5555* and make sure that your client app is serving by nginx.



d. For serving static assets of Front-End need to build client app, and add next configuration:

```
server {
   listen 8080;
   server_name static-app.net;

   root %path_to_your_client_app_folder%\dist\app;
   index index.html;

   location / {
        try_files $uri $uri/ /index.html =404;
   }
 }
```

Do not forget to restart nginx after changes to config file.

4. Implement Nginx configuration which will proxy requests from client app to REST API server so that API requests, which are made from client app's, are proxied and redirected to server's host and port. Eventually your application must seem working on one port. Two apps are managed by Nginx.

   a. Run your server, and make changes in your client app, so that all instead absolute URL of server, API request will be relative to client's host and port. Usually agreed to add */api/* prefix or other paths like that for the back-end service requests to distinguish them from the requests for static resources, i.e. for Angular app requesting products URL will be like: localhost:8080/api/products. After that save and rebuild your FE app.
   b. Following above changes, let's add a configuration for server to nginx.conf file:
      i. Under the first ***upstream*** section with client app, paste:

```
upstream rest_api_server {
      server localhost:3000; #provide port of your back-end's app
}
```

      ii. And in each ***server*** section, under the root location add config for back-end app:

```
location /api/ {
     proxy_pass http://rest_api_server/;

     proxy_http_version 1.1;
     proxy_set_header Host $host;
     proxy_set_header X-NginX-Proxy true;
     proxy_set_header X-Real-IP $remote_addr;
     proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
     proxy_set_header X-Forwarded-Proto $scheme;
 }
```

iii. Now check 5555 and 8080 ports in your browser and on both you should see working application with success API requests for products: one for local development and debugging (5555), and another for built apps with static files (8080).

5. **[optional]** Add *gzip compression* for static content.
6. Change the nginx.config file, so first root location is looking for client 's built static files, and if there is no such - proxy pass to running app. You will have something like:

```
location / {
  #Something will be here…
}

location @running_app {
  #And something will go here…
}
```

7. Configure *https connection* for both apps:
   a. If you do not have domain names that you own, you can modify your local hosts file to test your Nginx server blocks. It also will give you the ability to reach each site independently and setup secure connection. Open hosts file on your machine and add an entry:

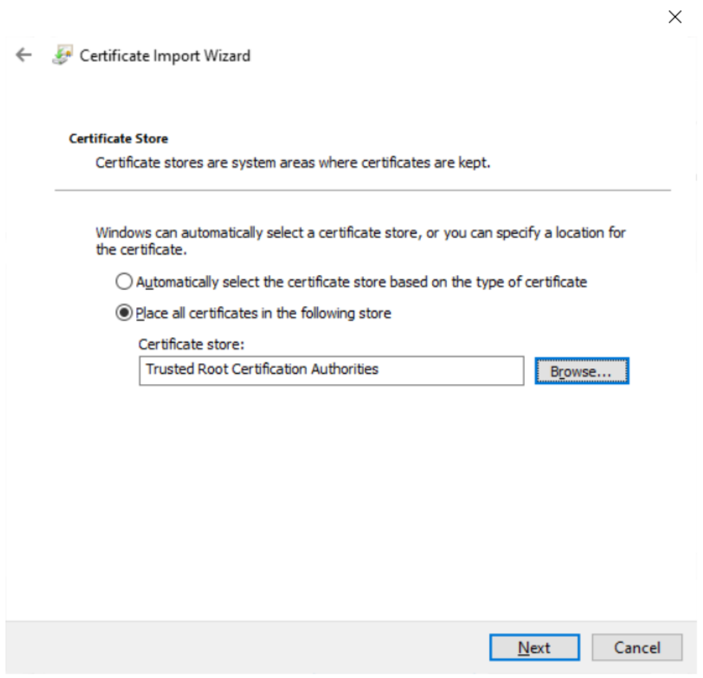
   ```
   127.0.0.1 devops-js-app.net #or any domain name you like
   ```

   b. Next step is to generate your self-signed certificate. To do this open terminal in folder where you will store your certificates (it can be any folder, even project folder) and run the following shortcut command (for Windows use Git Bash or Power Shell terminal, as in Command Prompt it will not work):

   ```
   openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes -keyout my-cert.key -out
   my-cert.crt -addext 'subjectAltName=DNS:devops-js-app.net,DNS:[your_IP_address]'
   ```

   Some settings may be stored in separate file and then used as input for the command. Refer course project repository's lab2 ssl folder for details. After generation need to install this certificate to make it trusted.

   For Windows: double click on file with .crt extension, in opened modal click on `Install Certificate…` button, next select current user and place all certificates in the following store


Certificate Import Wizard

Certificate Store

Certificate stores are system areas where certificates are kept.

Windows can automatically select a certificate store, or you can specify a location for the certificate.

( ) Automatically select the certificate store based on the type of certificate

(●) Place all certificates in the following store

Certificate store: Trusted Root Certification Authorities

Click next and finish the installation, agreed on next appeared modal.

c. To use SSL, the server should have the SSL certificate installed. As we have already completed it, time to change nginx configuration. Add new server section to the nginx config as showed below, and ensure the paths to *.crt and *.key files match to the location you saved them on:

```
server {
        listen 443 ssl http2;
        listen [::]:443 ssl http2;
        server_name devops-js-app.net; #your domain name from hosts file

        ssl_certificate %path_to_your_cert_folder%/my-cert.crt;
        ssl_certificate_key %path_to_your_cert_folder%/my-cert.key;

        location / {
                proxy_pass http://127.0.0.1:8080/;
        }
}
```

Also add another server block, that will redirect all URLs came to http to https:

```
server {
  listen 80;
  listen [::]:80;
  server_name devops-js-app.net;

  location / {
     return 301 https://$host$request_uri;
  }
}
```

d. Finally, go to the browser, and go to the *http://devops-js-app.net* and check whether you've been redirected to https. In case you'll have some troubles with certificates, try to clear cache, in the Application tab clear all stored data, or restart browser.

Part 5. Starting and serving BE & FE apps on remote OS.

1. Make sure that on remote OS installed nvm, node and nginx:
   ```
   sudo ufw app list
   ```
   a. For installing nvm:
      ```
      sudo apt install curl

      curl https://raw.githubusercontent.com/creationix/nvm/master/install.sh | bash

      source ~/.profile
      ```
      And check that installation successful:
      ```
      nvm --version
      ```
   b. For installing node:
      ```
      nvm install [node_lts_version]

      node --version
      ```
   c. For installing nginx:
      ```
      sudo apt install nginx -y
      ```
      Run next command and check whether it in list
      ```
      sudo ufw app list
      ```
      To make sure service is installed and running without any errors run:
      ```
      systemctl status nginx
      ```
      If, by some chance, the service is not active, use the following command to activate it:
      ```
      sudo systemctl enable nginx –now
      ```
      Next, enable Nginx in HTTP (Port 80) and HTTPS (Port 443):
      ```
      sudo ufw allow 'Nginx FULL'
      ```
2. Using cli/shell tools and SCP *(NOTE: here is windows analogue. Here and here are some useful topics about Secure Copy tool)* tool upload your **built apps** (server and client) files, package.json from server app, nginx config, certificates, etc. to Ubuntu OS:
   ```
   scp -Cr [server folder]/dist package.json ubuntu-sshuser: /var/app/[server folder name]

   scp -Cr [client folder]/dist/* [file1] [file2]  ubuntu- sshuser: /var/www/[client folder name]
   ```
   Note: if the folders do not exist in the destination, you should create them firstly and set rights for allowing your user to perform operations or change the owner group for created folders. To handle all these actions in one place – better to create script for simplicity and place all needed commands there (see example script in attachments), so when some source files changed you may easily rebuild the apps and update them on VM.

3. Once copying of files is done, need to setup apps in the Ubuntu:
   a. For server need to be installed all required dependencies from package.json:
      ```
      cd /var/app/[server folder name] && npm install
      ```

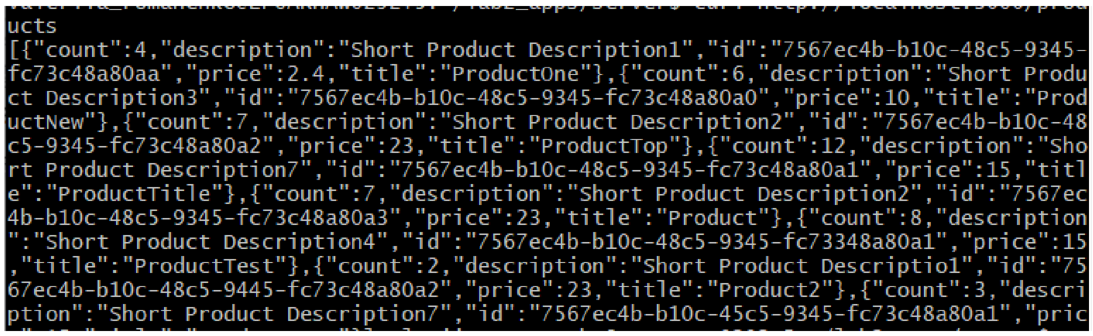
And start server:

npm run start:pm2

Check, that all is works and server handling requests with use of curl:

curl http://localhost:3000/products

If all is done and server works correctly - you should see json with products:



```
ucts
[{"count":4,"description":"Short Product Description1","id":"7567ec4b-b10c-48c5-9345-
fc73c48a80aa","price":2.4,"title":"ProductOne"},{"count":6,"description":"Short Produ
ct Description3","id":"7567ec4b-b10c-48c5-9345-fc73c48a80a0","price":10,"title":"Prod
uctNew"},{"count":7,"description":"Short Product Description2","id":"7567ec4b-b10c-48
c5-9345-fc73c48a80a2","price":23,"title":"ProductTop"},{"count":12,"description":"Sho
rt Product Description7","id":"7567ec4b-b10c-48c5-9345-fc73c48a80a1","price":15,"titl
e":"ProductTitle"},{"count":7,"description":"Short Product Description2","id":"7567ec
4b-b10c-48c5-9345-fc73c48a80a3","price":23,"title":"Product"},{"count":8,"description
":"Short Product Description4","id":"7567ec4b-b10c-48c5-9345-fc73348a80a1","price":15
,"title":"ProductTest"},{"count":2,"description":"Short Product Descriptio1","id":"75
67ec4b-b10c-48c5-9445-fc73c48a80a2","price":23,"title":"Product2"},{"count":3,"descri
ption":"Short Product Description7","id":"7567ec4b-b10c-45c5-9345-fc73c48a80a1","pric
```

b. For setup client you need to open the nginx.config and change the root and cert paths accordingly to their placement inside Ubuntu:

   cd /var/app/[client folder name] && nano [nging conf file name].conf

   Then copy nginx.config file to the */etc/nginx/sites-available/* folder:

   sudo cp ./[nging conf file name].conf /etc/nginx/sites-available/[your_domain]

   The next step is to activate nginx.config by creating a symbolic link between the *sites-available/* directory and the *sites-enabled/* directories:

   sudo ln -s /etc/nginx/sites-available/[your_domain] /etc/nginx/sites-enabled/[your_domain]

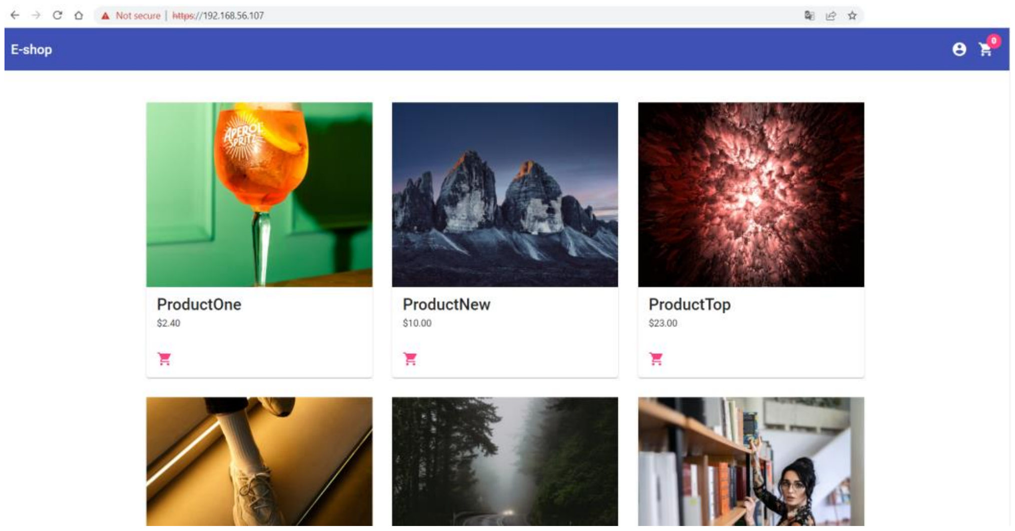
   To ensure that on localhost is displayed the correct page, you can delete the default nginx server block, as it will be called by default and override your nginx.config:

   sudo rm /etc/nginx/sites-enabled/default

   The last step is restarting nginx.

4. Finally, now you should be able to see your apps running on **80** or **443** port of instance. *(NOTE: use Host-only IP address of VM instance)*. In your host OS open browser and got to the your Ubuntu machine's IP and observe your application:

   http://[your_Ubuntu's_VM_IP]



5. Organize a flow (using shell scripts or provide README with instructions) of apps re-deployment after code is changed.