Task I

Main: (Web Server - Laravel Project - Config Details)
Sub: (Virtualization - Linux Os Setup - Git)

۱-web server: گرفتن درخواست HTTP از کاربر، گرفتن پاسخ از مقصد و تحویل دادن به کاربر
بهترین ها: Apache - Nginx

اپدیت OS:

Command: **$sudo apt update**

Apache:

Installation : **$sudo apt install apache2**

Checking web server : **$sudo systemctl status apache2**

Adjusting the firewall: **$sudo ufw allow 'Apache'**

با وارد کردن "localhost" یا IP دستگاه در مرورگر صفحه دیفالت اپاچی را میبینیم

Managing:

to **Stop** web server: **$sudo systemctl stop apache2**

to **Start** web server: **$sudo systemctl start apache2**

to **Restart** web server: **$sudo systemctl restart apache2**

to **Reload** web server: **$sudo systemctl reload apache2**

By default, Apache is configured to start automatically when the server boots

Disable this behavior: **$sudo systemctl disable apache2**

Re-enable the service: **$sudo systemctl enable apache2**

Setting up virtual hosts:
When using the Apache web server, you can use virtual hosts (similar to server blocks in Nginx) to encapsulate configuration details and host more than one domain from a single server. We will set up a domain called **your_domain**, but you should **replace this with your own domain name**.Apache on Ubuntu 22.04 has one server block enabled by default that is configured to serve documents from the `/var/www/html` directory. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying `/var/www/html`, create a directory structure within `/var/www` for a **your_domain** site, leaving `/var/www/html` in place as the default directory to be served if a client request doesn't match any other sites.

1.Create the directory for **your_domain:**
$sudo mkdir /var/www/your_domain

2.assign ownership of the directory to the user you're currently signed in as with the `$USER` environment variable:
$sudo chown -R $USER:$USER /var/www/your_domain

3.The permissions of your web root should be correct if you haven't modified your umask value, which sets default file permissions:
$sudo chmod -R 755 /var/www/your_domain

4.create a sample `index.html` page using `nano` or your favorite editor:
$sudo nano /var/www/your_domain/index.html

5.Add sample HTML

6.In order for Apache to serve this content, it's necessary to create a virtual host file with the correct directives. Instead of modifying the default configuration file located at `/etc/apache2/sites-available/000-default.conf` directly, make a new one at `/etc/apache2/sites-available/your_domain.conf`:
$sudo nano /etc/apache2/sites-available/your_domain.conf

7.Add in the following configuration block, which is similar to the default, but updated for your new directory and domain name:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName your_domain
    ServerAlias www.your_domain
    DocumentRoot /var/www/your_domain
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

8.enable the file with the `a2ensite` tool:

$sudo a2ensite your_domain.conf

9.Disable the default site defined in `000-default.conf`:

$sudo a2dissite 000-default.conf

10.test for configuration errors:

$sudo apache2ctl configtest

11.Restart Apache to implement your changes:

$sudo systemctl restart apache2

Step 6 – Getting Familiar with Important Apache Files and Directories

Now that you know how to manage the Apache service itself, you should take a few minutes to familiarize yourself with a few important directories and files.

Content

- `/var/www/html`: The actual web content, which by default only consists of the default Apache page you saw earlier, is served out of the `/var/www/html` directory. This can be changed by altering Apache configuration files.

Server Configuration

- `/etc/apache2`: The Apache configuration directory. All of the Apache configuration files reside here.
- `/etc/apache2/apache2.conf`: The main Apache configuration file. This can be modified to make changes to the Apache global configuration. This file is responsible for loading many of the other files in the configuration directory.
- `/etc/apache2/ports.conf`: This file specifies the ports that Apache will listen on. By default, Apache listens on port `80` and additionally listens on port `443` when a module providing SSL capabilities is enabled.
- `/etc/apache2/sites-available/`: The directory where per-site virtual hosts can be stored. Apache will not use the configuration files found in this directory unless they are linked to the `sites-enabled` directory. Typically, all server block configuration is done in this directory and then enabled by linking to the other directory with the `a2ensite` command.
- `/etc/apache2/sites-enabled/`: The directory where enabled per-site virtual hosts are stored. Typically, these are created by linking to configuration files found in the `sites-available` directory with the `a2ensite`. Apache reads the configuration files and links found in this directory when it starts or reloads to compile a complete configuration.
- `/etc/apache2/conf-available/`, `/etc/apache2/conf-enabled/`: These directories have the same relationship as the `sites-available` and `sites-enabled` directories but are used to store configuration fragments that do not belong in a virtual host. Files in the `conf-available` directory can be enabled with the `a2enconf` command and disabled with the `a2disconf` command.
- `/etc/apache2/mods-available/`, `/etc/apache2/mods-enabled/`: These directories contain the available and enabled modules, respectively. Files ending in `.load` contain fragments to load specific modules, while files ending in `.conf` contain the configuration for those modules. Modules can be enabled and disabled using the `a2enmod` and `a2dismod` commands.

Server Logs

- `/var/log/apache2/access.log`: By default, every request to your web server is recorded in this log file unless Apache is configured to do otherwise.
- `/var/log/apache2/error.log`: By default, all errors are recorded in this file. The `LogLevel` directive in the Apache configuration specifies how much detail the error logs will contain.

Nginx:

Installation: **$sudo apt install nginx**

Adjusting the firewall: **$sudo ufw allow 'Nginx HTTP’**

Checking the web server: **$systemctl status nginx**

با وارد کردن "localhost" یا IP دستگاه در مرورگر صفحه دیفالت Nginx را میبینیم

Managing:

to **Stop** web server: **$sudo systemctl stop nginx**

to **Start** web server: **$sudo systemctl start nginx**

to **Restart** web server: **$sudo systemctl restart nginx**

to **Reload** web server: **$sudo systemctl reload nginx**

By default, Nginx is configured to start automatically when the server boots. If this is not what you want, you can disable this behavior by typing:

Disable this behavior: **$sudo systemctl disable nginx**

Re-enable the service: **$sudo systemctl enable nginx**

Setting up server blocks:
When using the Nginx web server, server blocks (similar to virtual hosts in Apache) can be used to encapsulate configuration details and host more than one domain from a single server. We will set up a domain called **your_domain**, but you should **replace this with your own domain name**. Nginx on Ubuntu 22.04 has one server block enabled by default that is configured to serve documents out of a directory at `/var/www/html`. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying `/var/www/html`, let's create a directory structure within `/var/www` for our **your_domain** site, leaving `/var/www/html` in place as the default directory to be served if a client request doesn't match any other sites.

1.Create the directory for **your_domain** as follows, using the `–p` flag to create any necessary parent directories:

$sudo mkdir -p /var/www/your_domain/html

2.assign ownership of the directory with the `$USER` environment variable:

$sudo chown -R $USER:$USER /var/www/your_domain/html

3.The permissions of your web roots should be correct if you haven't modified your `umask` value, which sets default file permissions:

$sudo chmod -R 755 /var/www/your_domain

4.create a sample `index.html` page using `nano` or your favorite editor:

$nano /var/www/your_domain/html/index.html

5.Add sample HTML

6.In order for Nginx to serve this content, it's necessary to create a server block with the correct directives. Instead of modifying the default configuration file directly, let's make a new one at `/etc/nginx/sites-available/your_domain`:

$sudo nano /etc/nginx/sites-available/your_domain

7.Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:

```
server {
        listen 80;
        listen [::]:80;

        root /var/www/your_domain/html;
        index index.html index.htm index.nginx-debian.html;

        server_name your_domain www.your_domain;

       location / {
        try_files $uri $uri/ =404;
        }
}
```

8.enable the file by creating a link from it to the `sites-enabled` directory, which Nginx reads from during startup:

$sudo ln -s /etc/nginx/sites-available/your_domain /etc/nginx/sites-enabled/

9.Two server blocks are now enabled and configured to respond to requests based on their `listen` and `server_name` directives (you can read more about how Nginx processes these directives here):

- your_domain: Will respond to requests for your_domain and www.your_domain.
- default: Will respond to any requests on port 80 that do not match the other two blocks.

To avoid a possible hash bucket memory problem that can arise from adding additional server names, it is necessary to adjust a single value in the /etc/nginx/nginx.conf file. Open the file:

$sudo nano /etc/nginx/nginx.conf

10.Find the server_names_hash_bucket_size directive and remove the # symbol to uncomment the line. If you are using nano, you can quickly search for words in the file by pressing CTRL and w.

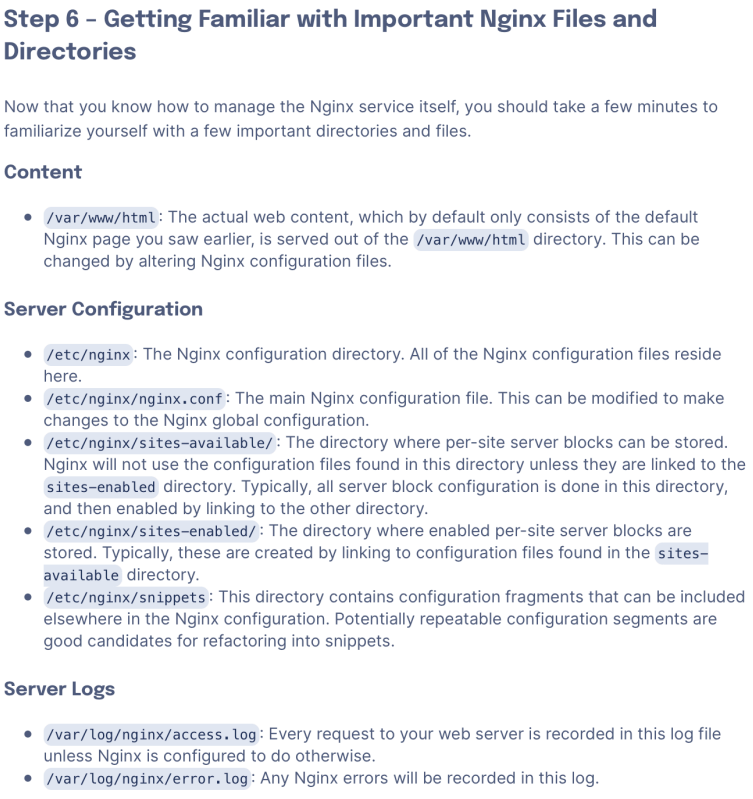
server_names_hash_bucket_size 64;

11.test to make sure that there are no syntax errors in any of your Nginx files:

$sudo nginx -t

12.restart Nginx to enable your changes:

$sudo systemctl restart nginx



Step 6 – Getting Familiar with Important Nginx Files and Directories

Now that you know how to manage the Nginx service itself, you should take a few minutes to familiarize yourself with a few important directories and files.

Content

- `/var/www/html`: The actual web content, which by default only consists of the default Nginx page you saw earlier, is served out of the `/var/www/html` directory. This can be changed by altering Nginx configuration files.

Server Configuration

- `/etc/nginx`: The Nginx configuration directory. All of the Nginx configuration files reside here.
- `/etc/nginx/nginx.conf`: The main Nginx configuration file. This can be modified to make changes to the Nginx global configuration.
- `/etc/nginx/sites-available/`: The directory where per-site server blocks can be stored. Nginx will not use the configuration files found in this directory unless they are linked to the `sites-enabled` directory. Typically, all server block configuration is done in this directory, and then enabled by linking to the other directory.
- `/etc/nginx/sites-enabled/`: The directory where enabled per-site server blocks are stored. Typically, these are created by linking to configuration files found in the `sites-available` directory.
- `/etc/nginx/snippets`: This directory contains configuration fragments that can be included elsewhere in the Nginx configuration. Potentially repeatable configuration segments are good candidates for refactoring into snippets.

Server Logs

- `/var/log/nginx/access.log`: Every request to your web server is recorded in this log file unless Nginx is configured to do otherwise.
- `/var/log/nginx/error.log`: Any Nginx errors will be recorded in this log.

۲-Laravel: محبوب‌ترین فریم ورک php

Install and Configure Laravel with Nginx

1.Installing Required PHP modules:

$sudo apt install php-mbstring php-xml php-bcmath php-curl

2.Creating a Database for the Application:

1.log in to the MySQL console as the root database user with:

$sudo mysql

2.To create a new database, run the following command from your MySQL console:

CREATE DATABASE travellist;

3.create a new user and grant them full privileges on the custom database you've just created. In this example, we're creating a user named travellist_user with the password password, though you should change this to a secure password of your choosing:

CREATE USER 'travellist_user'@'%' IDENTIFIED WITH mysql_native_password BY 'password';

4.give this user permission over the travellist database:

GRANT ALL ON travellist.* TO ‘travellist_user'@'%';

5.exit the MySQL shell:

exit

6.test if the new user has the proper permissions by logging in to the MySQL console again, this time using the custom user credentials:

$mysql -u travellist_user -p

7.confirm that you have access to the travellist database:

SHOW DATABASES;

8.create a table named places in the travellist database. From the MySQL console, run the following statement:

```
CREATE TABLE travellist.places (
id INT AUTO_INCREMENT,
name VARCHAR(255),
visited BOOLEAN,
PRIMARY KEY(id)
);
```

9.populate the places table with some sample data:

```
INSERT INTO travellist.places (name, visited)
VALUES ("Tokyo", false),
("Budapest", true),
("Nairobi", false),
("Berlin", true),
("Lisbon", true),
("Denver", false),
("Moscow", false),
("Olso", false),
("Rio", true),
("Cincinnati", false),
("Helsinki", false);
```

10.confirm that the data was successfully saved to your table, run:

```
SELECT * FROM travellist.places;
```

11.exit the MySQL console:

```
exit
```

3. New Laravel Project:

1. Creating one:

You will now create a new Laravel application using the composer create-project command. This Composer command is typically used to bootstrap new applications based on existing frameworks and content management systems. Throughout this guide, we'll use travellist as an example application, but you are free to change this to something else. The travellist application will display a list of locations pulled from a local MySQL server, intended to demonstrate Laravel's basic configuration and confirm that you're able to connect to the database.

1.go to your user's home directory:

$cd ~

2.The following command will create a new `travellist` directory containing a barebones Laravel application based on default settings:

$composer create-project --prefer-dist laravel/laravel travellist

3.access the application's directory and run Laravel's `artisan` command to verify that all components were successfully installed:

$cd travellist

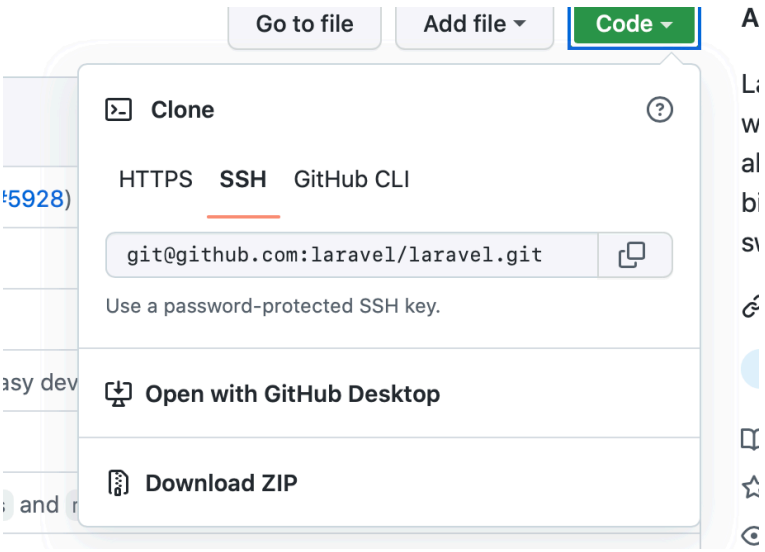
$php artisan

2. Getting from Github

1.go to destination directory

2.open project in GitHub

3.get ssh from this part (detail of making ssh key in the end)



4. Clone the project:

$git clone git@github.com:laravel/laravel.git

4. Configuring Laravel

The Laravel configuration files are located in a directory called `config`, inside the application's root directory. Additionally, when you install Laravel with Composer, it creates an environment file. This file contains settings that are specific to the current environment the application is running, and will take

precedence over the values set in regular configuration files located at the `config` directory. Each installation on a new environment requires a tailored environment file to define things such as database connection settings, debug options, application URL, among other items that may vary depending on which environment the application is running.

We'll now edit the `.env` file to customize the configuration options for the current application environment.

1.Open the `.env` file using your command line editor of choice. Here we'll use `nano`:

$nano .env

Even though there are many configuration variables in this file, you don't need to set up all of them now. The following list contains an overview of the variables that require immediate attention:

- APP_NAME: Application name, used for notifications and messages.
- APP_ENV: Current application environment.
- APP_KEY: Used for generating salts and hashes, this unique key is automatically created when installing Laravel via Composer, so you don't need to change it.
- APP_DEBUG: Whether or not to show debug information at client side.
- APP_URL: Base URL for the application, used for generating application links.
- DB_DATABASE: Database name.
- DB_USERNAME: Username to connect to the database.
- DB_PASSWORD: Password to connect to the database.

By default, these values are configured for a local development environment that uses Homestead, a prepackaged Vagrant box provided by Laravel. We'll change these values to reflect the current environment settings of our example application.

In case you are installing Laravel in a development or testing environment, you can leave the APP_DEBUG option enabled, as this will give you important debug information while testing the application from a browser. The APP_ENV variable should be set to development or testing in this case.

In case you are installing Laravel in a production environment, you should disable the APP_DEBUG option, because it shows to the final user sensitive information about your application. The APP_ENV in this case should be set to production.

The following .env file sets up our example application for development:

APP_NAME= TravelList
APP_ENV= development

APP_KEY= APPLICATION_UNIQUE_KEY_DONT_COPY
APP_DEBUG= true
APP_URL=http://domain_or_IP

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=travellist
DB_USERNAME=travellist_user
DB_PASSWORD=password

Adjust your variables accordingly. When you are done editing, save and close the file to keep your changes. If you're using nano, you can do that with CTRL+X, then Y and Enter to confirm.
Your Laravel application is now set up, but we still need to configure the web server in order to be able to access it from a browser. In the next step, we'll configure Nginx to serve your Laravel application.

To test database connection and everything go to project directory and run this:

$sudo php artisan migrate

5.Setting Up Nginx

We have installed Laravel on a local folder of your remote user's home directory, and while this works well for local development environments, it's not a recommended practice for web servers that are open to the public internet. We'll move the application folder to `/var/www`, which is the usual location for web applications running on Nginx.

1.use the `mv` command to move the application folder with all its contents to `/var/www/travellist`:

$sudo mv ~/travellist /var/www/travellist

2.Now we need to give the web server user write access to the storage and cache folders, where Laravel stores application-generated files:

$sudo chown -R www-data.www-data /var/www/travellist/storage

$sudo chown -R www-data.www-data /var/www/travellist/bootstrap/cache

3.The application files are now in order, but we still need to configure Nginx to serve the content. To do this, we'll create a new virtual host configuration file at /etc/nginx/sites-available:

$sudo nano /etc/nginx/sites-available/travellist

4.The following configuration file contains the recommended settings for Laravel applications on Nginx:

```
server {
        listen 80;
        server_name server_domain_or_IP;
        root /var/www/travellist/public;

        add_header X-Frame-Options “SAMEORIGIN”;
        add_header X-XSS-Protection "1; mode=block”;
        add_header X-Content-Type-Options “nosniff";

        index index.html index.htm index.php;

        charset utf-8;

        location / {
                try_files $uri $uri/ /index.php?$query_string;
        }

        location = /favicon.ico { access_log off; log_not_found off;
}
        location = /robots.txt  { access_log off; log_not_found
off; }

        error_page 404 /index.php;

        location ~ \.php$ {
                fastcgi_pass unix:/var/run/php/php8.1-fpm.sock;
                fastcgi_index index.php;
                fastcgi_param SCRIPT_FILENAME
$realpath_root$fastcgi_script_name;
                include fastcgi_params;

        }
```

```
        location ~ /\.(?!well-known).* {
            deny all;
        }
    }
```

Copy this content to your /etc/nginx/sites-available/travellist file and, if necessary, adjust the highlighted values to align with your own configuration. Save and close the file when you're done editing.

5.To activate the new virtual host configuration file, create a symbolic link to travellist in sites-enabled:

$sudo ln -s /etc/nginx/sites-available/travellist /etc/nginx/sites-enabled/

6.To confirm that the configuration doesn't contain any syntax errors, you can use:

$sudo nginx -t

7.To apply the changes, reload Nginx with:

$sudo systemctl reload nginx

8.Now go to your browser and access the application using the server's domain name or IP address, as defined by the server_name directive in your configuration file:

http://server_domain_or_IP

Installing composer:

1.Installing PHP and Additional Dependencies:

In addition to dependencies that should be already included within your Ubuntu 22.04 system, such as git and curl, Composer requires php-cli in order to execute PHP scripts in the command line, and unzip to extract zipped archives. We'll install these dependencies now.

run the following command to install the required packages:

$sudo apt install php-cli unzip

2.Downloading and Installing Composer

Composer provides an installer script written in PHP. We'll download it, verify that it's not corrupted, and then use it to install Composer.
Make sure you're in your home directory, then retrieve the installer using curl:

$cd ~
$curl -sS https://getcomposer.org/installer -o /tmp/composer-setup.php

Next, we'll verify that the downloaded installer matches the SHA-384 hash for the latest installer found on the Composer Public Keys / Signatures page. To facilitate the verification step, you can use the following command to programmatically obtain the latest hash from the Composer page and store it in a shell variable:

$HASH=`curl -sS https://composer.github.io/installer.sig`

If you want to verify the obtained value, you can run:

$echo $HASH

Now execute the following PHP code, as provided in the Composer download page, to verify that the installation script is safe to run:

$php -r "if (hash_file('SHA384', '/tmp/composer-setup.php') === '$HASH') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"

If the output says `Installer corrupt`, you'll need to download the installation script again and double check that you're using the correct hash. Then, repeat the verification process. When you have a verified installer, you can continue.
To install composer globally, use the following command which will download and install Composer as a system-wide command named composer, under /usr/local/bin:

$sudo php /tmp/composer-setup.php —install-dir=usr/local/bin —filename=composer

To test your installation, run:

$composer

Source

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-composer-on-ubuntu-22-04

https://www.digitalocean.com/community/tutorials/how-to-install-the-apache-web-server-on-ubuntu-22-04
https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-22-04
https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-laravel-with-nginx-on-ubuntu-22-04

Ssh key:https://kinsta.com/blog/generate-ssh-key/
https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Fastcgi: https://www.nginx.com/resources/wiki/start/topics/examples/fastcgiexample/
https://www.digitalocean.com/community/tutorials/understanding-and-implementing-fastcgi-proxying-in-nginx

proxy_pass:https://www.digitalocean.com/community/tutorials/how-to-configure-nginx-as-a-web-server-and-reverse-proxy-for-apache-on-one-ubuntu-20-04-server
https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/
http://nginx.org/en/docs/http/ngx_http_proxy_module.html