Application web, et connexion à l'API OPENAI

Introduction:

Durant mon stage de Mai à Juin 2023, au journal Jeune Afrique, j'ai réalisé une application web:

Cette application est codée en PhP, HTML et CSS, et est conteneurisée à partir de Docker. Elle nous permet d'interagir avec l'API OPEN AI, créée par les créateurs de Chat GPT, et nous permet d'intégrer un modèle de langage avancé, permettant de générer du texte cohérent et pertinent en fonction des instructions fournies. Elle permet donc d'obtenir des réponses à nos questions, sans pour autant devoir passer par l'interface de ChatGPT.

Dans notre cas, l'application a été pensée pour accompagner les journalistes lors du processus de création d'articles, de titres et la recherche d'informations. Elle fournit ainsi une assistance clé dans la rédaction, et permet ainsi d'intégrer chatGPT directement dans les outils des journalistes pour gagner du temps tout en ajoutant une logique pour personnaliser les réponses de ChatGPT dans notre contexte du journalisme.

Installation et configuration:

Docker:

Ici le lien vers la documentation officielle de Docker:
https://docs.docker.com/engine/install/

Nos commandes seront réalisées sur Linux, avec la distribution Ubuntu 22.04.3, commençons par mettre à jour nos paquets:

```
stephane@OptiPlex-7070:~$ sudo apt update
```

Nous devons réaliser une suite de commandes permettant d'installer des pré-requis au bon fonctionnement de docker, apache et php:

-Installation de paquets nécessaires à l'installation de Docker:

```
stephane@OptiPlex-7070:~$ sudo apt install apt-transport-https ca-certificates c
url software-properties-common
Lecture des listes de paquets... Fait
```

-Ajout des dépôts Docker, en commençant par la clé GPG:

```
stephane@OptiPlex-7070:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
[sudo] Mot de passe de stephane :
```

-Ajout du dépôt Docker:

```
stephane@OptiPlex-7070:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
Dépôt : « deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable »
Description :
Archive for codename: focal components: stable
Plus d'informations : https://download.docker.com/linux/ubuntu
Ajout du dépôt.
Appuyez sur [ENTRÉE] pour continuer ou Ctrl-c pour annuler
Found existing deb entry in /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list
Found existing deb-src entry in /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list
Atteint :1 http://fr.archive.ubuntu.com/ubuntu jammy InRelease
```

Nous mettons ensuite à jour les paquets, avec cette fois-ci, les paquets de Docker:

```
stephane@OptiPlex-7070:~$ sudo apt update
Atteint :1 http://fr.archive.ubuntu.com/ubuntu jammy InRelease
Atteint :2 https://download.docker.com/linux/ubuntu focal InRelease
Réception de :3 https://download.docker.com/linux/ubuntu jammy InRelease [48,8 kB]
Atteint :4 http://fr.archive.ubuntu.com/ubuntu jammy-updates InRelease
Atteint :5 http://packages.microsoft.com/repos/code stable InRelease
Atteint :6 http://fr.archive.ubuntu.com/ubuntu jammy-backports InRelease
Réception de :7 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
159 ko réceptionnés en 1s (185 ko/s)
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
```

On s'assure de bien installer Docker depuis son dépôt, et non pas depuis le dépôt Ubuntu:

```
stephane@OptiPlex-7070:~$ apt-cache policy docker-ce
docker-ce:
  Installé : 5:24.0.7-1~ubuntu.22.04~jammy
  Candidat : 5:24.0.7-1~ubuntu.22.04~jammy
 Table de version :
 *** 5:24.0.7-1~ubuntu.22.04~jammy 500
```

Ici on peut remarquer qu'une version est déjà installé, ayant déjà réalisé ces commandes, j'ai déjà installé Docker, dans le cas où il ne serait pas installé, à la place de la version de Docker, : "Installed : (none)" serait écrit à la place.

Ainsi, il ne nous reste plus qu'à installer Docker! :

```
stephane@OptiPlex-7070:~$ sudo apt install docker-ce
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
docker-ce est déjà la version la plus récente (5:24.0.7-1~ubuntu.22.04~jammy).
```

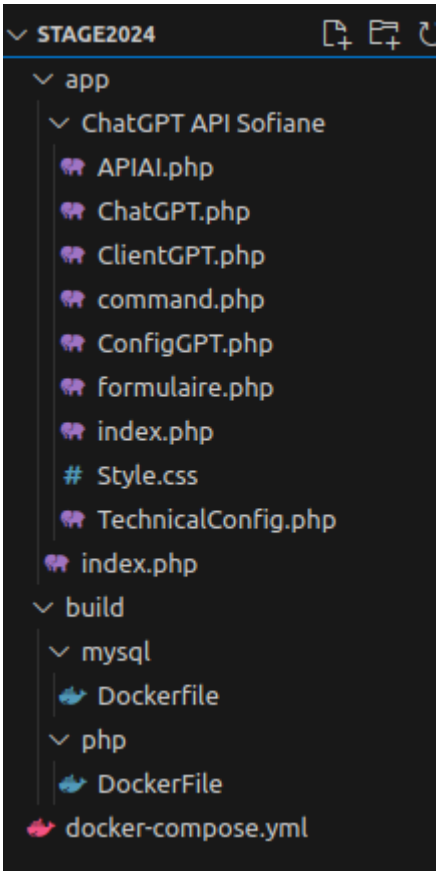
Ici il est bien précisé que j'ai déjà installé Docker, cette commande devrait permettre d'installer la version de Docker la plus récente par rapport à votre version d'ubuntu.

Pour "finaliser" l'installation de Docker, il ne nous reste plus qu'à vérifier le statut, et confirmer qu le service est actif:

```
stephane@OptiPlex-7070:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2024-01-11 09:12:34 CET; 1h 40min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 1394 (dockerd)
      Tasks: 13
     Memory: 84.9M
        CPU: 2.244s
     CGroup: /system.slice/docker.service
             └─1394 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

janv. 11 09:12:33 OptiPlex-7070 dockerd[1394]: time="2024-01-11T09:12:33.923753522+01:00" le
janv. 11 09:12:34 OptiPlex-7070 dockerd[1394]: time="2024-01-11T09:12:34.084442734+01:00" le
janv. 11 09:12:34 OptiPlex-7070 dockerd[1394]: time="2024-01-11T09:12:34.140119496+01:00" le
janv. 11 09:12:34 OptiPlex-7070 dockerd[1394]: time="2024-01-11T09:12:34.140475930+01:00" le
janv. 11 09:12:34 OptiPlex-7070 dockerd[1394]: time="2024-01-11T09:12:34.340307612+01:00" le
janv. 11 09:12:34 OptiPlex-7070 systemd[1]: Started Docker Application Container Engine.
janv. 11 10:41:37 OptiPlex-7070 dockerd[1394]: time="2024-01-11T10:41:37.187682433+01:00" le
janv. 11 10:41:37 OptiPlex-7070 dockerd[1394]: time="2024-01-11T10:41:37.187735383+01:00" le
janv. 11 10:41:37 OptiPlex-7070 dockerd[1394]: time="2024-01-11T10:41:37.187681672+01:00" le
janv. 11 10:41:37 OptiPlex-7070 dockerd[1394]: time="2024-01-11T10:41:37.187792702+01:00" le
lines 1-22/22 (END)
```

Nous allons réaliser une application web, possédant plusieurs pages, dont un formulaire, et accessible via le localhost.

Une arborescence est nécessaire pour garantir la bonne mise en place, et l'optimisation de notre application (d'un point de vue organisationnel).



Ici vous pouvez voir que dans mon dossier "*Stage2024*" j'ai créé 2 dossiers : "*app*" et "*build*"; le dossier "*app*"contient le code de l'application, et le dossier "*build*" contient les fichiers DockerFile, nécessaire à la configuration de Docker, et qui détaillent les procédures d'installations pour les conteneurs.

Dans mon exemple ci-contre, il est a noté que j'ai également installé MySQL, mais cela ne nous intéresse pas pour notre application Web.

Enfin, le fichier "*docker-compose-yml*" configure Docker avec les services précises, et permet de gérer l'application et de la "construire", en effet à chaque lancement de notre application web dans docker, il sera nécessaire de "construire" ou "build" notre fichier "*docker-compose.yml*", et ce dernier sera chargé de déployer l'application.

Attardons-nous sur les différents fichiers permettant le bon fonctionnement de Docker.

Docker-compose.yml:

```yaml
version: "3.9"
services:
  php-apache:
    ports:
      - "80:80"
    build: './build/php'
    volumes:
      - ./app/ChatGPT API Sofiane:/var/www/html
```

- On spécifie la version du docker compose (outil spécifiant le processus de définition), ici **"3.9"**
- On définit ensuite un service nommé, ici : **"php-apache"**,
- **"ports"**: Spécifie que notre application sera disponible depuis le port 80 de notre machine, mappé au port 80 du conteneur.

- **“build”**: Indique le chemin pour accéder au DockerFile concerné depuis le dossier courant, ici dans le répertoire “*php*”, situé dans le répertoire “*build*” où le chemin est, depuis le répertoire courant: ***./build/php.***
- **“volumes”**: Cette section permet de partager des données entre le conteneur et l’hôte (nous !). Ainsi, cette ligne permet de mapper le répertoire ***/app/ChatGPT API Sofiane*** avec le répertoire ***/var/www/html*** du conteneur, et donc permettre à l’application d’apparaître sur la page localhost de notre machine. De même que, toute modifications apportées dans le répertoire hôte apparaîtront également dans le répertoire du conteneur.
    - Ici, le volume **“app”** est déclaré à la fin du programme.

Pour résumer sur le fichier Docker Compose, il nomme un service “php-apache”, et spécifie les paramètres lui étant associés, paramètres nécessaires au bon fonctionnement de la conteneurisation.

DockerFile:

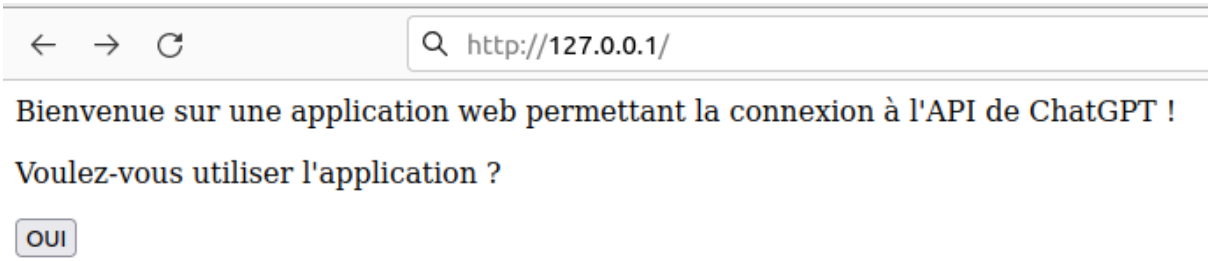
```
FROM php:8.1-apache

RUN apt-get update && \
    docker-php-ext-install mysqli pdo pdo_mysql
```

- **“FROM” :** Spécifie à Docker l'image à utiliser. Ici, on prend l'image officielle de php-apache.
- **“RUN”:** Commande utilisée pour exécuter des commandes dans le DockerFile, ici on met à jour la liste de paquets (+ d'installation de paquets utiles pour la connexion à la base de données).

Nous en avons terminé avec les fichiers nécessaires à la configuration de Docker et de ses conteneurs.
Il ne nous reste plus qu’à lancer les conteneurs, afin de tester le résultat, et pour cela nous devons nous rendre à la racine de notre site (ici Stage2024/), et lancer la commande: *sudo docker compose up*

```
stephane@OptiPlex-7070:~$ cd Stage2024/
stephane@OptiPlex-7070:~/Stage2024$ sudo docker compose up
[sudo] Mot de passe de stephane :
[+] Running 2/0
 ✔ Container stage2024-php-apache-1  Created                    0.0s
 ✔ Container stage2024-mysql-1       Crea...                    0.0s
```

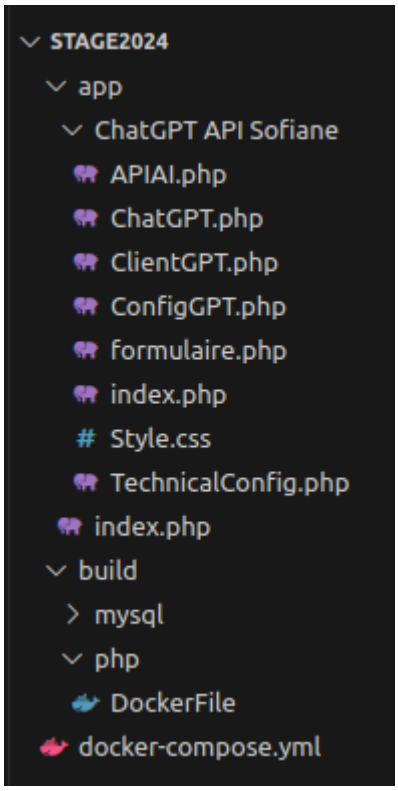
Et si on ouvre le navigateur pour se rendre à l’adresse : http://127.0.0.1 ou à http://localhost, on arrivera sur la page contenant le projet(ATTENTION ! Il est essentiel d’avoir des fichiers php dans le dossier app, dans mon cas dans le dossier ‘app/ChatGPT API Sofiane’; avec un fichier index.php, qui s’affichera):



Notre application est fonctionnelle ! Et la conteneurisation avec Docker a fonctionnée !

Structure de l'application :

Architecture de l'application:



Les fichiers contenant le code de notre application se situent dans le dossier ***'app/ChatGPT API Sofiane'***, (depuis la racine de l'application).

Ici on retrouve:

- ***"index.php"***: Ici ce fichier nous sert de porte d'entrée, de page d'accueil, et sert concrètement à nous rediriger vers la page du formulaire:

```
print( string $arg ): int
<?ph
    print("Bienvenue sur une application web permettant la connexion à l'API de ChatGPT !");
?>
    <form action="formulaire.php" method="post">
        <p>Voulez-vous utiliser l'application ?</p>
        <p> <input type="submit" value="OUI"> </p>
```

- ***“formulaire.php”***: Ici, le but de cette page est de récupérer la question qui sera entrée, pour ensuite la stocker dans la variable $POST[‘demande’], et pourra ainsi être utilisée dans notre code, au fichier **“APIAI.php”** plus tard.

Entrez une question à poser à l'API de ChatGPT :

OK

- ***“APIAI.php”***: Ce fichier joue le rôle de “chef d’orchestre”, il fait appel à

```
<form action="APIAI.php" method="post">
    <h1>
        <p>Entrez une question à poser à l'API de ChatGPT : <input type="text" name="demande" /></p>
        <p> <input type="submit" value="OK"> </p>
    </h1>
```

tous les fichiers nécessaires au fonctionnement de l’application; nous reviendrons sur celui-ci après avoir présenté les classes et fichiers qui sont appelés dans ce fichier.

- ***“TechnicalConfigGPT.php”***: Cette classe est chargée de la **configuration purement technique** pour l'utilisation de l'API GPT.

On définit dans le constructeur la clé d’API GPT (ici caché car chaque clé est unique et sensible);

```
class TechnicalConfigGPT
{

    public function __construct(private string $key="**************************************************")
    {}

    public function head()
    {
        $headers = array(
            "Content-Type: application/json",
            "Authorization: Bearer ". $this->key,
        );
        return $headers;
    }
}
```

La méthode head(), renverra un tableau contenant les en-tête HTTP nécessaires pour une requête vers l’API GPT(on note que la clé mentionnée dans le constructeur est ajoutée dans l’en-tête d'autorisation de la requête, et sert de jeton d’authentification).

- ***"ConfigGPT.php"***: Cette classe est également destinée à gérer un aspect de la configuration lié à l'utilisation de l'API GPT (les paramètres de la réponse) (config métier).

```php
class ConfigGPT
{

    public function __construct(private string $model = "gpt-3.5-turbo-instruct", private int $temperature = 0, p
    {}

    public function data($prompt)
    {
        $data = array(
            "model" => $this->model,
            "prompt" => $prompt,
            "temperature" => $this->temperature,
            "max_tokens" => $this->max_tokens,
            "top_p" => $this->top_p,
            "frequency_penalty" => $this->frequency_penalty,
            "presence_penalty" => $this->presence_penalty,
        );
        return $data;
    }
}
```

La méthode ***"data($prompt)"*** prend en argument une variable, dans laquelle la question posée à l'étape du formulaire se trouvera. De nombreux paramètres nécessaires et utiles pour pouvoir contrôler la réponse de l'API GPT sont également définis. On peut noter certains paramètres que nous utiliserons tout particulièrement:

- ★ ***"model":*** définir le modèle GPT à utiliser (traite et génère le texte)
- ★ ***"prompt":*** La chaîne de caractères(la question que l'on pose à l'API GPT) qui sera traité, et spécifié dans la requête
- ★ ***"temperature":*** Cette propriété contrôle le niveau de créativité de ChatGPT. Une Température faible (proche de 0) engendre des réponses plus conservatrices et répétitives, tandis qu'une température élevée (proche de 1) produit des réponses plus variées et créatives.

Cette méthode "***data***" renvoie un tableau associatif regroupant des données à envoyer à l'API GPT via une requête.

- ***"ClientGPT.php"***: Cette classe est dédiée à la création d'une session avec l'API GPT, via des requêtes HTTP, les méthodes "***data***" et "***head***" seront appelées dans cette classe, dans la méthode : "***session***".

Ici, la méthode "***session***" prend deux variables en paramètres, un tableau, ainsi qu'un tableau associatif. Il s'agit des deux tableaux renvoyés par les fonctions "***data***" et "***head***".

```php
class ClientGPT
{
    public function session($data, $headers)
    {
        $curl = curl_init();
    // Configurer les options de la requête cURL
    curl_setopt_array($curl, array(
        CURLOPT_URL => "https://api.openai.com/v1/completions",
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_ENCODING => "",
        CURLOPT_MAXREDIRS => 10,
        CURLOPT_TIMEOUT => 30,
        CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
        CURLOPT_CUSTOMREQUEST => "POST",
        CURLOPT_POSTFIELDS => json_encode($data),
        CURLOPT_HTTPHEADER => $headers,
    ));
    $response = curl_exec($curl);
    $answer = json_decode($response,true);
    curl_close($curl);
    return $answer;
    }
}
```

"***session($data, $headers)***", les paramètres représentent les données, et l'en-tête de la requête envoyé à l'API GPT.

Cette méthode utilise "***curl***", une bibliothèque PHP permettant d'envoyer une requête HTTP, avec comme URL cible celle de l'API GPT : "**https://api.openai.com/v1/completions**", elle permet également de spécifier certains paramètres (les paramètres de base sont suffisant pour notre application)

La requête est effectuée avec "***curl_exec***", la réponse à cette requête est alors stockée dans la variable :"***$response***", au format JSON ! Il est alors nécessaire d'utiliser la fonction "***json_decode***", afin d'obtenir une réponse sous forme de tableau associatif.

Enfin, cette méthode retourne la réponse, sous forme de tableau associatif, d'où il est possible d'extraire la chaîne de caractère correspondant à la réponse, de l'API GPT, vis à vis de la question initialement transmise via la méthode "**data**" de ConfigGPT.

- ***"ChatGPT.php"***: ***ChatGPT*** est une classe faisant appel aux classes ***ClientGPT***, ***TechnicalConfigGPT*** et ***ConfigGPT*** via le constructeur.

La classe ChatGPT sert **de point d'entrée pour faciliter l'utilisation de l'application.**

```php
require_once(__DIR__.'/ConfigGPT.php');
require_once(__DIR__.'/ClientGPT.php');
require_once(__DIR__.'/TechnicalConfig.php');

class ChatGPT
{

    public function __construct(private ConfigGPT $config, private ClientGPT $client, private TechnicalConfigGPT $TechConfig)
    {
    }

    public function send($prompt)
    {
        $headers = $this->TechConfig->head();
        $data = $this->config->data($prompt);
        $answer = $this->client->session($data, $headers);
        return $answer;
    }
}
```

La méthode "***send($prompt)***", prend en paramètre le texte d'entrée pour l'API GPT et elle utilise les différentes méthodes des classes précédemment définis:

- L'appel à la méthode ***head()***, permet d'obtenir l'en-tête de la requête possédant le jeton d'authentification,
- L'appel de la méthode ***data($prompt)***, permet d'obtenir les données nécessaire à l'envoi de la requête (possédant le texte entré),
- L'appel de la méthode ***session($data, $headers)*** permet d'effectuer la requête HTTP vers l'API GPT, on stock alors la réponse à cette requête dans la variable $answer.

Ainsi, la réponse de la requête à l'API GPT est renvoyée par la méthode ***send*** de cette classe.

- ***"APIAI.php":***

```php
require_once(__DIR__.'/ChatGPT.php');
require_once(__DIR__.'/ConfigGPT.php');
require_once(__DIR__.'/ClientGPT.php');
require_once(__DIR__.'/TechnicalConfig.php');

if(isset($_POST['demande'])){
    $reponse_neutre="Rédige la réponse à cette question de façon neutre, comme le ferait un journaliste, sans opinions personnelles:".$_POST['demande'];
    $config = new ConfigGPT();
    $TechConfig = new TechnicalConfigGPT();
    $client = new ClientGPT();
    $ChatGPT = new ChatGPT($config, $client, $TechConfig);
    $answer=$ChatGPT->send($reponse_neutre);
    echo "<pre>";
    print("<p>La question est : ".htmlspecialchars($_POST['demande'])." ? </p><hr>");
    $answer["choices"][0]['text']=str_replace(". ", ".<br>", $answer["choices"][0]['text']);
    print($answer["choices"][0]['text']);
    echo "</pre>";
}
else{
    print("Vous n'avez pas posé de question.\n Redirection vers la page d'acceuil de cette application.");
}
?>
<form action="formulaire.php" method="post">
        <p> <input type="submit" value="OUI"> </p>
```

Tout d’abord, on inclut les différents fichiers des classes ***ChatGPT***, ***ClientGPT***, ***TechnicalConfigGPT*** et ***ConfigGPT,*** avec:
“require_once(__DIR__.'/Nom_Fichier.php');”,

- ➢ On vérifie l'existence d’une question dans la variable ***$POST[‘demande’]***, si “***isset($POST[‘demande’])==1***”, alors on entre dans la boucle générant une requête et une réponse de l’API GPT, sinon un message d’erreur est affiché.

- ➢ Une fois dans la boucle, on instancie les classes ***ChatGPT***, ***ClientGPT***, ***TechnicalConfigGPT*** et ***ConfigGPT,*** respectivement dans les variables : ***$ChatGPT***, ***$client***, ***$TechConfig*** et ***$config.***

- ➢ On instancie la variable $ChatGPT avec les trois autres instances de classes :
  “***$ChatGPT = new ChatGPT($config, $client, $TechConfig)***”
- ➢ La réponse à notre demande initiale, est stockée sous forme de tableau associatif dans la variable ***$answe***r, et prendra la valeur de retour de la méthode ***send()*** de l’instance de classe ***ChatGPT***.
- ➢ Il ne nous reste plus qu’à afficher la réponse émise par l’API GPT, et pour cela nous devons afficher l’élément situé à l’indice: ***[‘choices’][0][‘text’]*** du tableau associatif obtenu.

```
La question est : Qui a inventé Linux ? ?
```

```
Selon les sources historiques, Linux a été créé par Linus Torvalds, un étudiant finlandais en informatique, en 1991.
Cependant, le système d'exploitation open-source a été développé en collaboration avec de nombreux contributeurs et est continuellement amélioré par une communauté de programmeurs à travers le monde.
```

- ***“style.css”***: Ce fichier contient le code css servant à la mise en forme des différents éléments du site (couleur, taille, bordures,..)