# Présentation du projet

Vous allez devoir, par groupe de 3, développer un serveur web.

## Critères d'évaluation

- **il doit être publié sur github**
  - il doit posséder un readme qui explique brièvement ce que fait le projet et **comment l'utiliser**
- lancer le projet doit se résumer à un .bat, un .sh ou quelques commandes documentées
- le serveur doit être implémenté en Java (ou en C/C++)
- il ne doit pas utiliser de librairie « toutes faites »
  - pas de jar en dépendance (en dehors de la jre bien sûr) => « *from scratch* »
  - mais peut utiliser quelques algorithmes tout faits comme base64/md5
- il doit savoir exploiter une requête GET en HTTP/1.x (en ignorant les entêtes non gérées)
  - découpage de la requête « à la main » (*parsing*) et traitement
- **le serveur doit être fonctionnel (i.e. afficher correctement les sites exemples fournis)**
- le serveur doit gérer proprement les erreurs les plus courantes (400, 404)
- il doit gérer plusieurs connexions en parallèle (en étant *par exemple* multi-tâche avec un thread = une connexion)
- **il doit être « bien écrit »** : code documenté, maintenable, évolutif
- il doit afficher du log sur *stdout* : ip de l'appelant + requête
- il doit gérer le multisite (i.e. pouvoir héberger plusieurs domaines)
- il doit pouvoir protéger une ressource par une authentification basique
  - un répertoire est protégé s'il contient un fichier .htpasswd
  - le fichier contient des lignes sous la forme « username:password_en_md5 »
- il faut externaliser la configuration dans un fichier *properties*
  - répertoire racine où mettre les sites web, port tcp (80 par défaut)

Il doit implémenter **une fonctionnalité « bonus » au choix** :

- gérer les server-side includes (#include et #exec)
- générer du contenu dynamique à l'aide d'un programme externe (php, python, node)
- gérer le listing des répertoires (doit pouvoir être désactivé par configuration)
- compresser les ressources js et css en gzip

## Bien écrit

Le code doit être **documenté** : les commentaires doivent être réguliers et pertinent, ne pas expliquer ce qui est fait mais pourquoi c'est fait.

Le code doit être **maintenable** : n'importe qui doit pouvoir intervenir facilement pour corriger le code au besoin. Le code doit être compréhensible.

Éviter par exemple :

```
if(entête == 'Content-Length') {
} else if(entête == 'Content-Type') {
} else if(entête == 'Accept') {
else {
        // ...
}
```

Le code doit être **évolutif** : pensez à combien de temps vous prendrait d'ajouter une fonctionnalité.

# Rappel sur le multi-tâche

Par défaut, un programme est démarré avec un seul thread, le thread principal (ou thread graphique dans le cas d'une application graphique).

Toute opération d'I/O (lecture d'un fichier, d'écoute du réseau, ...) bloque le thread principal et donc le programme en entier.

Il faut donc créer des thread supplémentaires pour effectuer ces tâches en parallèle.

Explications : https://www.irif.fr/~sighirea//cours/reseauxM/java.thread.html

Pour gérer les requêtes, on *peut* par exemple utiliser le *Design Pattern* Producteur/Consommateur.

Explications : http://www-igm.univ-mlv.fr/~forax/ens/java-avance/cours/pdf/13-Concurrence-Producteur-consommateur.pdf

Qu'est-ce qu'un Design Pattern ? https://fr.wikipedia.org/wiki/Patron_de_conception