



Répartition des rôles à 3

◆ Équipe Réseau & I/O

Responsable : Dev 1

- Mise en place des **sockets** (création, bind, listen, accept).
- Implémentation de la **boucle poll()/select()** → gestion multi-clients.
- Déconnexions propres et non-bloquant.
- Gestion des **buffers** (lecture/écriture).
- S'assurer qu'il n'y a jamais de blocage → robustesse/stress test.

👉 Doit fournir une API claire :

- `get_request(fd)`
 - `send_response(fd, response)`
-

◆ Équipe Parsing & HTTP

Responsable : Dev 2

- Parsing des requêtes HTTP (ligne de requête + headers + body).
- Gestion `Content-Length`, `Transfer-Encoding: chunked`.
- Validation + génération d'**erreurs HTTP** (`400`, `413`, `404`, `405`, etc.).
- Construction des **réponses HTTP** (status line, headers, body).
- Gestion des méthodes : `GET`, `POST`, `DELETE`.
- Génération d'**error pages par défaut**.

👉 Doit fournir :

- `Request` (struct/classe) avec `method`, `uri`, `headers`, `body`.
 - `Response` (classe) avec `status_code`, `headers`, `body`.
-

♦ Équipe Configuration & CGI

Responsable : Dev 3

- Écriture du **parser de fichier de config** inspiré de nginx.
 - `listen`, `root`, `index`, `error_page`, `client_max_body_size`, `location {}`, etc.
- Implémentation de la logique **mapping request -> configuration**.
- Implémentation du **CGI** :
 - `fork()` + `execve()`.
 - Passage des variables d'environnement.
 - Gestion stdin/stdout, EOF, Content-Length.
- Vérification des règles liées aux `location` (`upload_store`, redirection, autoindex).

👉 Doit fournir :

- `Config` (classe) qui peut être interrogée par `Server`.
 - `CGIHandler` qui exécute et renvoie la sortie formatée.
-



Coordination

- Chaque dev travaille dans son **module**, mais avec **interfaces propres**.
- **Point critique** : bien définir les **structures communes** (`Request`, `Response`, `Config`) pour éviter de recoder 3 fois la même chose.

- Tests unitaires par équipe avant intégration.
 - Intégration finale = fusion **Réseau (I/O) + HTTP Parser + Config/CGI**.
-



Répartition des charges

- **Dev 1 (Réseau)** : gros travail au début (poll, sockets), puis support/stabilité.
 - **Dev 2 (HTTP)** : cœur du serveur, doit travailler en parallèle une fois qu'un socket simple marche.
 - **Dev 3 (Config/CGI)** : parser et CGI sont indépendants, peut commencer très tôt.
-

👉 En résumé :

- **Dev 1 = Transport des données (réseau)**
 - **Dev 2 = Traduction HTTP (parser + réponse)**
 - **Dev 3 = Contexte serveur (config + CGI)**
-

Veux-tu que je t'organise ça sous forme de **planning chronologique à 3** (qui fait quoi à chaque phase de la roadmap que je t'ai donnée) ?

Vous avez dit :

je veux bien une version pdf de ce que tu viens de m'écrire s'il te plait

42GPT a dit :

It seems like I can't do more advanced data analysis right now. Please try again later.

Vous avez dit :

oui s'il te plait

42GPT a dit :

Parfait 👍

Comme je ne peux pas générer directement le PDF pour l'instant, je vais te donner le contenu prêt à copier-coller. Tu pourras ensuite le mettre en PDF facilement avec un éditeur de texte (LibreOffice, Word → "Exporter en PDF") ou via un simple [pandoc](#).

Répartition des rôles – Projet Webserv (3 personnes)

Équipe Réseau & I/O (Dev 1)

- Mise en place des sockets (creation, bind, listen, accept).
- Implémentation de la boucle `poll()/select()` → gestion multi-clients.
- Gestion des déconnexions propres et non-bloquant.
- Gestion des buffers (lecture/écriture).
- S'assurer qu'il n'y a jamais de blocage → robustesse/stress test.
- **API fournie :**
 - `get_request(fd)`
 - `send_response(fd, response)`

Équipe Parsing & HTTP (Dev 2)

- Parsing des requêtes HTTP (ligne de requête + headers + body).
- Gestion `Content-Length`, `Transfer-Encoding: chunked`.
- Validation + génération d'erreurs HTTP (`400`, `413`, `404`, `405`, etc.).
- Construction des réponses HTTP (status line, headers, body).
- Implémentation des méthodes : `GET`, `POST`, `DELETE`.
- Génération d'error pages par défaut.
- **Structures fournies :**

- `Request` (method, uri, headers, body)
 - `Response` (status_code, headers, body)
-

Équipe Configuration & CGI (Dev 3)

- Parser de fichier de configuration inspiré de nginx.
 - Directives : `listen`, `root`, `index`, `error_page`, `client_max_body_size`, `location`.
 - Mapping `request` → `configuration`.
 - Implémentation du CGI : `fork()` + `execve()`, passage des variables d'environnement, `stdin/stdout`.
 - Gestion EOF si pas de `Content-Length`.
 - Respect des règles `location` : `upload_store`, `redirection`, `autoindex`.
 - **Classes fournies :**
 - `Config`
 - `CGIHandler`
-

Coordination

- Chaque dev travaille dans son module avec interfaces propres.
 - Bien définir les **structures communes** (`Request`, `Response`, `Config`).
 - Tests unitaires par équipe avant intégration.
 - Intégration finale = **Réseau (I/O) + HTTP Parser + Config/CGI**.
-

Répartition des charges

- **Dev 1 (Réseau)** : gros travail au début (poll, sockets), puis stabilisation.
 - **Dev 2 (HTTP)** : cœur du serveur, en parallèle dès que les sockets marchent.
 - **Dev 3 (Config/CGI)** : parser et CGI indépendants, peut commencer très tôt.
-

Résumé

- **Dev 1 = Transport des données (réseau)**
- **Dev 2 = Traduction HTTP (parser + réponse)**
- **Dev 3 = Contexte serveur (config + CGI)**



Planning de travail Webserv à 3

Phase 1 – Squelette (Jour 1-2)

- **Dev1** : Crée le `Makefile`, initialise la base du projet (`src/`, `include/`, `conf/`).
 - **Dev2** : Définit les structures `Request` et `Response`.
 - **Dev3** : Pose le plan du parser de configuration (syntaxe inspirée de nginx).
-

Phase 2 – Socket & Poll (Jour 3-5)

- **Dev1** : Implémente un serveur simple avec `socket`, `bind`, `listen`, `accept`.
 - **Dev1** : Passe en non-bloquant (`fcntl`) et ajoute la boucle `poll()`.
 - **Dev2** : Prépare un `RequestParser` basique (juste lire la première ligne).
 - **Dev3** : Avance sur le parser du fichier de config (lecture des blocs `server {}`).
-

Phase 3 – Parsing HTTP (Jour 6-8)

- **Dev1** : Stabilise la gestion multi-clients (tests avec plusieurs connexions).
 - **Dev2** : Implémente parsing complet des headers (`Content-Length`, etc.).
 - **Dev3** : Termine une première version du parser de config (`listen`, `root`, `index`).
-

Phase 4 – Réponses simples (Jour 9-10)

- **Dev1** : Gère lecture/écriture dans les buffers côté client.
- **Dev2** : Implémente `Response` avec `200 OK`, `404 Not Found`.

- **Dev3** : Connecte la conf minimale avec le serveur (ex: `listen 8080, root`).
-

Phase 5 – Méthodes HTTP (Jour 11-13)

- **Dev1** : Continue robustesse côté poll (ajout/suppression de clients).
 - **Dev2** : Implémente `GET`, `POST`, `DELETE` (sans CGI encore).
 - **Dev3** : Étend la conf (`error_page`, `client_max_body_size`, `autoindex`).
-

Phase 6 – CGI (Jour 14-17)

- **Dev1** : Assure que `fork()` + `execve()` fonctionne sans bloquer poll.
 - **Dev2** : Ajoute gestion des réponses CGI dans `Response`.
 - **Dev3** : Implémente `CGIHandler` (stdin → CGI, stdout → HTTP response).
-

Phase 7 – Robustesse (Jour 18-20)

- **Dev1** : Stress tests (`ab -n 1000 -c 100`), gestion déconnexions.
 - **Dev2** : Vérifie conformité des status codes HTTP avec Nginx.
 - **Dev3** : Finalise parsing complet (redirections, `upload_store`, méthodes autorisées).
-

Phase 8 – Intégration & Nettoyage (Jour 21-22)

- Intégration finale : tous testent ensemble.
- Vérification avec `valgrind` (pas de leaks).

- Tests croisés : navigateurs, `curl`, comparaison avec Nginx.
-

Phase 9 – Bonus (si temps)

- **Dev1** : Sessions/cookies basiques.
 - **Dev2** : Support multi-CGI (php, python).
 - **Dev3** : Ajout des virtual hosts (`server_name`).
-

👉 Ce planning est pensé pour **3 semaines de travail** environ, en avançant en parallèle. Il permet à chaque dev d'avoir **un module clair**, tout en convergeant régulièrement.