CY Tech – PréING2 – Année 2025-2026 - MEF1

Rapport du Projet
C-Wildwater

21/12/2025 – Module: Informatique 3

Réalisé par :
- Qays Chaabaoui
- Firmin Godinou
- Ayman Madrassi

SOMMAIRE

1. Introduction

1.1 Description de l'équipe

1.2 Description du Sujet

2. Organisation du Projet et Flux de Travail

2.1 Organisation de l'équipe

2.2 Flux de travail

3. Problèmes Rencontrés et Solutions Apportées

4. Résultats Obtenus

4.1 *Analyse des performances des usines (Histo)*

4.2 Analyse des pertes d'eau (Leaks)

4.3 Interfaces de pilotage (Script Shell)

5. Conclusion

Introduction :

1.1 Description de l'équipe

Ce projet a été réalisé par notre équipe constitué de trois personnes : Ayman Madrassi, Firmin Godinou et Qays Chaabaoui. Du fait de notre complémentarité et nos compétences nous nous sommes investis afin concevoir un programme répondant aux exigences du cahier des charges. La répartition des taches s'est faite de manière intuitive nous permettant de répondre au mieux aux différents problèmes rencontré.

1.2 Description du sujet

L'application conçue pour le projet C-WildWater et développée en langage C et Script Shell vise à réaliser la synthèse de données d'un système de distribution d'eau. Le programme que nous devons concevoir doit permettre le traitement automatisé et l'analyse de volumes massifs de données détaillant la distribution hydraulique.
Notre approche se base sur un rassemblement technique permettant de filtrer et de traiter efficacement les informations contenues dans un fichier CSV volumineux (plus de 500 Mo), afin de suivre le parcours de l'eau à travers les usines de traitement et les zones de stockage, jusqu'aux consommateurs finaux.
Au cours de ce programme, l'utilisateur interagira par le biais d'une interface en ligne de commande gérée par un script Shell, qui permet d'organiser toutes les actions requises dans l'ordre souhaité. Cela comprend des requêtes telles que la création d'histogrammes de performance des usines (capacité, volume capté, volume traité), ainsi que le calcul exact du rendement de distribution pour déterminer les fuites d'eau sur le réseau aval. Le projet se concentre spécifiquement sur l'amélioration des temps de calcul et de l'empreinte en mémoire grâce à l'emploi de structures de données sophistiquées telles que les arbres AVL. Ainsi, l'objectif principal est de développer une application solide et efficace qui automatise l'analyse de la topologie du réseau et appuie la gestion des ressources hydriques.

2. Organisation du Projet et Flux de Travail

2.1 Organisation de l'équipe

Afin de réaliser ce projet, nous avons mis en place une organisation basée sur de nombreux piliers :

L'ensemble du projet est constitué d'un dépôt Git centraliser répertorier et hébergés sur GitHub. Notre utilisation de Git n'était pas optionnelle, c'était obligatoire.

- Chaque ajout, modification ou suppression de code faisait l'objet d'un commit avec un message décrivant l'action effectuée. Cela nous permettait de garder une trace sur notre évolution ainsi que de nous permettre d'éviter de reproduire les mêmes erreurs.

- Git nous a facilité le travail. Chaque membre pouvait travailler sur des fonctionnalités distinctes dans des branches séparées, nous permettant d'éviter des confusions ou la suppression de code.

- Le dépôt centralisé a joué un rôle de sauvegarde automatique de notre projet. En cas de problème matériel sur l'ordinateur d'un membre, le code restait accessible et protégé. De plus, cela nous garantissait que tous les membres du groupe travaillaient toujours sur la version la plus à jour du projet.

De plus, afin de faciliter la communication, nous avons eu recours à des outils tels que Discord afin que l'on puisse échanger de manière très simple.
- L'objectif était de créer un environnement où chaque membre se sentait à l'aise pour poser des questions, partager ses idées ou signaler des difficultés.
- Les décisions importantes ainsi que les clarifications sur le cahier des charges y étaient partagées afin que chacun d'entre nous puissent suivre l'avancer globale du projet.

2.2 Flux de travail

Afin atteindre les objectifs définis dans le cahier des charges de manière structurée et progressive, nous avons adopté un flux de travail répétitif. Nous avons développé le programme progressivement en instaurant constamment des retours et en nous adaptant aux défis rencontrés. Notre travail a été divisé en plusieurs phases que nous avons répété pour chaque fonctionnalité du projet. Nous avons tout d'abord planifié, analysé et conceptualisé les exigences du programme ainsi que les attendus spécifiques de chaque fonctionnalité. Puis, nous débutions le code mais pas de manière globale. Que ce soit Qays, Firmin ou Ayman nous devions chacun développé de spécifique et pas à pas en incrémentant petit à petit chaque partie du code. Nous réalisions différents tests au fur et à mesure du projet afin de savoir quelle partie du code pourrait posséder d'éventuel problème que ce soit au niveau des donnée ou encore entre chaque incrémentation. Enfin, l'étape qui a été pour nous la plus importante a été la relecture et l'optimisation constante du code car cela nous permettait de constamment faire le lien avec le cahier des charges et ainsi limiter au maximum le risques d'oublier n'importe quelle consigne.

3. Problèmes Rencontrés et Solutions Apportées

Tout d'abord, la gestion du fichier de données CSV s'est révélé plus compliquée que prévu à cause de son format et de sa taille. Comme les lignes ne sont pas triées et contiennent parfois des valeurs manquantes, notre programme plantait souvent lors de la lecture. On a dû mettre en place une lecture ligne par ligne robuste avec des vérifications strictes pour ignorer ou traiter correctement les données incomplètes sans arrêter le programme brutalement.

Ensuite, la réalisation du script Shell a été un défi car c'est lui qui pilote tout le projet. On a eu du mal à gérer correctement tous les arguments possibles (comme histo ou leaks) et à traiter les cas d'erreurs, par exemple si l'utilisateur oublie un paramètre. Il fallait aussi s'assurer que le script lance automatiquement la compilation du programme C via le Makefile si l'exécutable n'était pas présent. Finalement, on a réussi à structurer le script pour qu'il vérifie bien les codes de retour du programme C et qu'il affiche la durée d'exécution à la fin.

Enfin, l'implémentation des arbres AVL imposée par le sujet a constitué la difficulté technique majeure. On n'avait pas l'habitude de manipuler des structures qui s'auto-équilibrent, et nos premières versions ne faisaient pas les rotations correctement, ce qui rendait la recherche d'usines trop lente. On a dû passer beaucoup de temps à revoir la logique des rotations gauche

et droite pour garantir que la complexité reste logarithmique et ainsi respecter les contraintes de performance.

4. Résultats Obtenus

4.1 *Analyse des performances des usines (Histo)*

Nous avons mis en place une fonction qui permet de produire des histogrammes. Le logiciel C analyse le fichier CSV, crée un arbre AVL des usines et compile les informations en fonction du critère sélectionné : capacité maximale, volume capté ou volume traité. Les informations sont transférées vers un fichier temporaire, ensuite notre script Shell commande Gnuplot pour produire les graphiques des 10 usines les plus grandes et 50 les plus petites.

4.2 Analyse des pertes d'eau (Leaks)

Notre application facilite le calcul du volume d'eau perdu en aval d'une usine donnée, dans le cadre de la gestion des fuites. L'utilisateur soumet un numéro d'identification d'installation (par exemple : « Complexe d'installation #RH400057F »). Le programme élabore un arbre de répartition et effectue un parcours récursif du réseau jusqu'aux utilisateurs afin d'additionner les pertes (leaks). Les résultats sont conservés dans un fichier d'historique, et si l'usine n'existe pas, le programme retourne bien la valeur -1.

4.3 Interfaces de pilotage (Script Shell)

Nous avons élaboré un script Shell fiable qui fait office de point d'entrée unique. Il contrôle l'existence des arguments (fichier CSV, commande histo ou fuites). Il procède automatiquement à la compilation du programme C via “make” si le fichier exécutable n'est pas présent. À la fin de l'exécution, il présente la durée totale du traitement en millisecondes, nous donnant la possibilité d'évaluer l'efficacité de nos améliorations.

5. Conclusion

Le projet C-WildWater nous a apporté de nombreuses leçons, non seulement en ce qui concerne le langage C et l'algorithmique avancée (arbres AVL, graphes), mais également sur l'architecture logicielle combinant Shell et C. La gestion de la mémoire et le traitement d'un fichier de 500 Mo nous paraissaient compliqués, mais nous avons persévéré. Nous

avons réussi à élaborer un programme efficace, qui respecte les limitations de temps et de mémoire imposées. Nous sommes satisfaits du résultat : l'application marche parfaitement, elle offre une analyse efficace du réseau hydraulique et l'interface textuelle est facile à utiliser. Nous avons renforcé nos aptitudes en Git et en collaboration d'équipe.