



Soutenance 2

S4-B1

Logiciel de compression

*Lucas PERES
Timéo BERTHAULT
Aboubakar CHARF*

Avril 2025

Table des matières

1	Introduction	3
2	Présentation du groupe	4
2.1	Lucas PERES	4
2.2	Aboubakar CHARF	4
2.3	Timeo BERTHAULT	4
3	Organisation	5
3.1	Répartition	5
3.2	Prévision et avancement	5
3.2.1	Prévision	5
3.2.2	Avancement	5
4	Aspects techniques	6
4.1	Interface	6
4.2	Compression/Décompression tar.gz	9
4.2.1	Compression	11
4.2.2	Décompression	12
4.3	Compression Image/vidéo	13
4.3.1	Algorithmie	13
4.3.2	Problèmes rencontrés	14
4.4	Site Web	16
4.4.1	Améliorations graphiques	16
5	Conclusion	18
6	Sources	20

1 Introduction

Le projet réalisé par notre groupe est un logiciel de compression de fichiers. La compression de fichier est la réduction du nombre de bits nécessaires pour représenter des données. Compresser les données permet d'optimiser la capacité de stockage et la vitesse de transfert des fichiers.

Comme son nom l'indique, l'objectif du projet est de pouvoir compresser des fichiers de différents types (images et vidéos essentiellement mais également des dossiers, des fichiers textes...) en étant le plus efficace possible.

Le terme "efficace" varie selon le type du fichier mais pour une image ou une vidéo par exemple il s'agirait de maintenir au maximum la qualité d'image du média une fois traité.

2 Présentation du groupe

2.1 Lucas PERES

Ayant toujours été attiré par l'informatique, j'ai rejoins Epita en sortant du lycée, mon premier gros projet en équipe s'est déroulé l'année dernière, ce projet est pour moi l'occasion d'étoffer mes compétences et d'apprendre de nouvelles choses mais également rattraper l'échec qu'a été personnellement le projet OCR.

2.2 Aboubakar CHARF

Je m'appelle Aboubakar Charf, 21 ans, j'ai expérincé les précédents projets avec foi et vaillance (à tel point que c'est mon deuxième projet S4 que je vais réaliser ce semestre). On peut dire que j'ai toujours été attiré par l'informatique à partir du lycée mais que cela n'a pas toujours été réciproque. Mais cela ne m'empêche pas de toujours traiter mon rapport au travail avec passion et détermination, car ma philosophie me l'interdit. J'ai donc hate d'aboutir à un projet final digne de ce que nous avions prévus avec mes camarades et amis du groupe de projet S4VjEnjoyers.

2.3 Timeo BERTHAULT

Après une Terminale avec une spécialité en Mathématiques et une option en Physique-chimie avec une orientation Mathématiques approfondies, j'ai choisi de poursuivre mes études à l'EPITA. N'ayant pas choisi la spécialité NSI, je me retrouve sans réel bagage informatique avant d'entrer à l'EPITA, et ce projet sera donc ma deuxième expérience après celui de l'année dernière.

3 Organisation

3.1 Répartition

Tâches	Responsable
Compression	
Compression/Décompression de fichiers	Lucas
Compression/Décompression vidéo et image	Timéo
Affichage	
Interface utilisateur	Timéo/Lucas
Liaison programme/interface	Timéo/Aboubakar
Interface personnelle	Aboubakar
Site web	
Site	Aboubakar
Téléchargement	Aboubakar
(BONUS) Compression depuis le site web	Lucas

TABLE 1 – Répartition des tâches

3.2 Prévision et avancement

3.2.1 Prévision

Soutenances	Fichiers	Vidéos	Interface	Site Web
1	30%	30%	35%	40%
2	60%	60%	60%	70%
3	100%	100%	100%	100%

Nous avons organisé une répartition plus ou moins équilibrée des tâches en fonction des difficultés possibles, et nous ferons de notre mieux pour la respecter.

3.2.2 Avancement

Soutenance	Fichiers	Vidéos	Interface	Site Web
2	70%	70%	90%	70%

4 Aspects techniques

4.1 Interface

J'avais commencé à faire l'interface qui permet de choisir un fichier afin de le compresser ou le décompresser. Pour cela j'ai créer deux boutons différents : un pour la compression et l'autre pour la décompression.

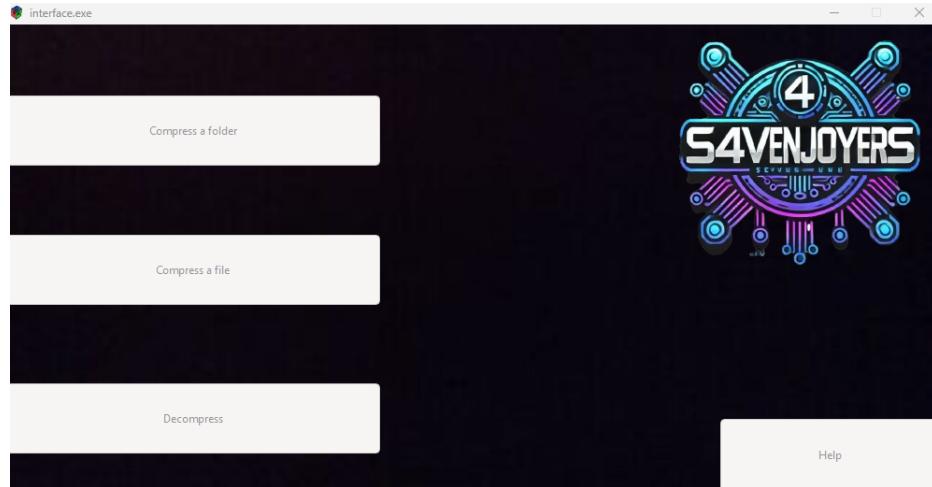
Le bouton pour la décompression appelle une fonction qui permet de naviguer dans les fichiers de l'ordinateur afin de choisir un fichier compressé. Une fois le fichier sélectionné, le but est de pouvoir naviguer dans celui-ci afin de choisir les fichiers à extraire.

Le bouton pour la compression appelle lui aussi une fonction qui permet de naviguer dans les fichiers de l'ordinateur, sauf que lui ne retourne que le chemin du fichier choisi afin de pouvoir appliquer la fonction qui fait la compression à ce fichier.

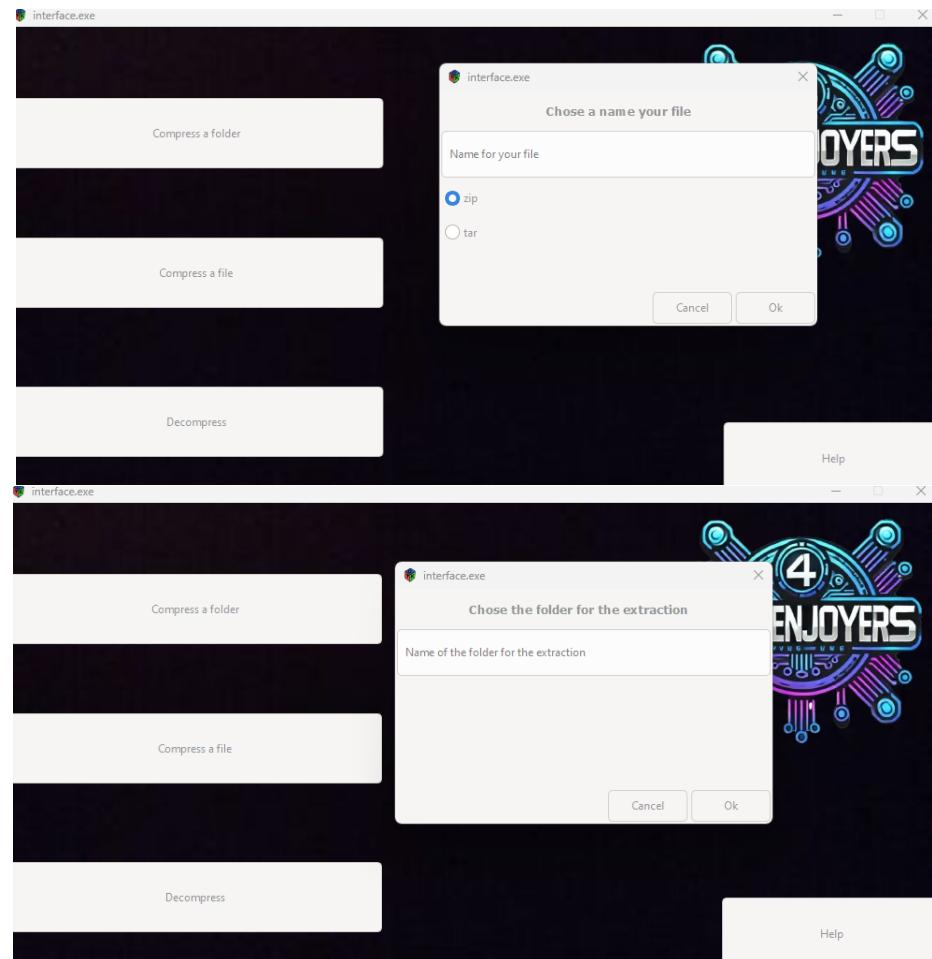
Pour cette soutenance, j'avais pour objectif de régler les problèmes présents dans l'interface (affichage des fichiers et chemin correct) avant de relier les fonctions de compression et décompression aux boutons afin de pouvoir n'utiliser que l'interface sans passer par le terminal. Et bien sur, commencer à rendre l'interface plus esthétique, en ajoutant des couleurs et changeant l'agencement des widgets.

J'ai donc commencé par régler les problèmes présents. Je suis donc passé sur glade (Comme mentionné pour la première soutenance) afin de pouvoir ensuite gérer le visuel de l'interface plus facilement et j'ai résolu les problèmes de chemins que j'avais.

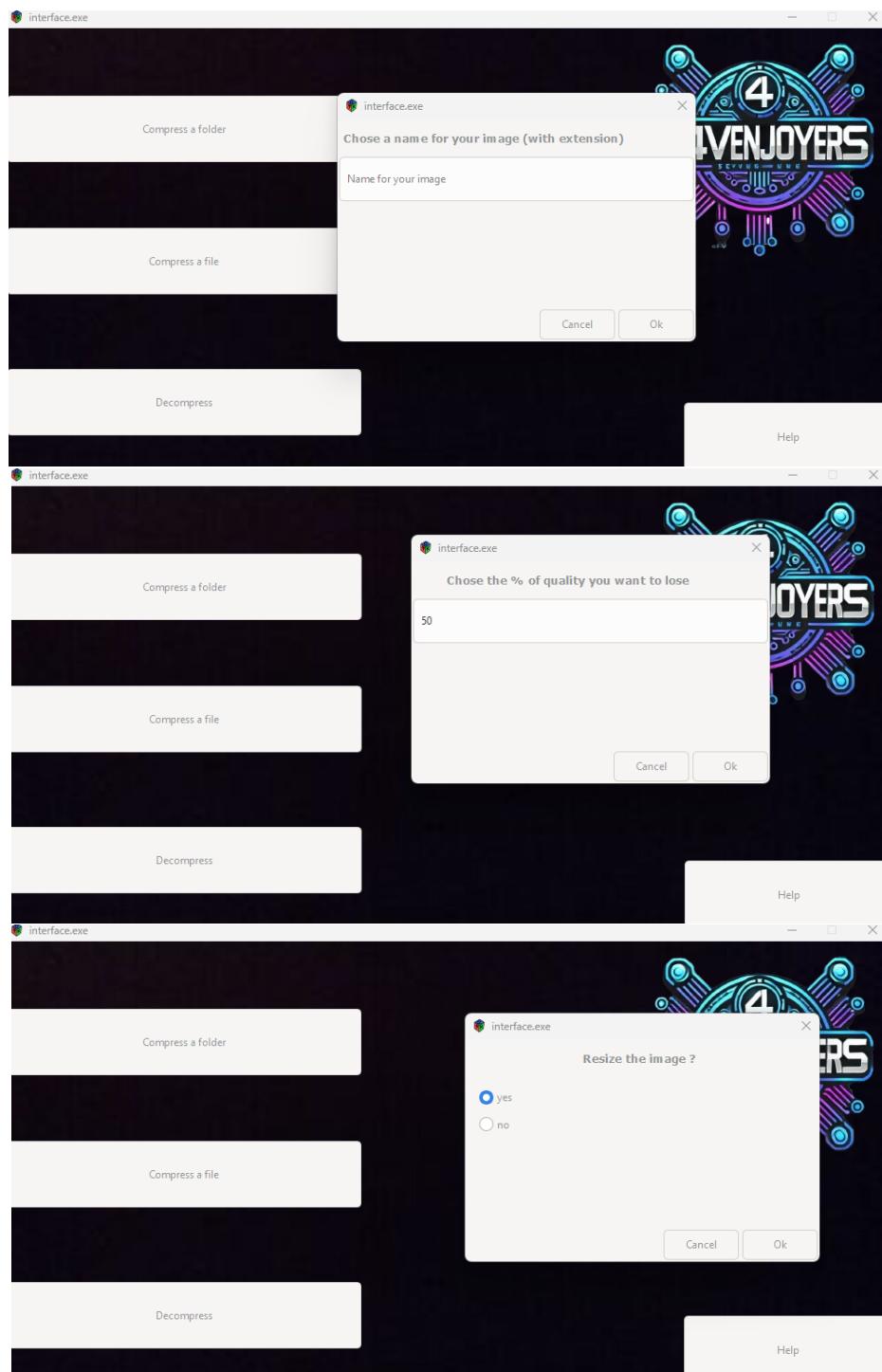
Tout d'abord, afin de rendre l'interface plus agréable visuellement, j'ai réarrangé les boutons et ajouté un fond noir ainsi que le logo du groupe. Ça peut paraître simple mais le tout rend l'interface vraiment plus esthétique comme vous pouvez le voir ci-dessous.



Comme prévu, j'ai relié les boutons aux fonctions de compression et de décompression. Pour cela j'ai ajouté un bouton afin que l'utilisateur choisisse s'il veut compresser un dossier ou juste un fichier. De plus, j'ai ajouté une fenêtre afin que l'utilisateur puisse choisir le nom du fichier compressé créé mais aussi choisir le type (zip et tar pour l'instant) conformément aux fonctions créées par mes camarades. Pour la décompression, l'utilisateur peut choisir le nom du dossier dans lequel l'extraction se fera.



Ensuite j'ai fais plusieurs fenêtres pour gérer les cas des images. Lorsque l'utilisateur choisit de compresser une image l'interface le détecte et ainsi demande à l'utilisateur de nommer l'image avec l'extension, puis en fonction du type de l'image désirée deux différentes fenêtres peuvent s'afficher (la fonction est expliquée plus en détails dans la section 4.3 Compression Image).



J'ai eu des problèmes lorsque j'ai voulu appeler des GtkDialog car lorsque je faisais mes recherches sur internet ils ne me donnaient pas les bonnes fonctions

d'appel. Cependant, une fois les bonnes fonctions trouvées, étant donnée que j'ai principalement utilisé des GtkDialog l'implémentation de l'interface à été beaucoup plus rapide. Il y avait aussi le fait que lorsque je lançais un dialog et que je le fermais je ne pouvais plus le réouvrir car je le fermais définitivement. Enfin, c'est assez compliqué de trouver des bons tutoriels car la plupart sont en C ou en python et les fonctions ne sont donc pas les mêmes. Il était donc nécessaire pour moi de chercher dans la documentation rust gtk car les tutoriels ne m'étaient pas vraiment utiles.

Pour la prochaine soutenance, j'ai pour but de totalement finir l'interface. Pour cela, j'essayerai d'ajouter des fenêtres d'erreurs (par exemple lorsque qu'on appuye sur ok alors qu'aucun fichier n'est selectionné) ainsi que d'affiner l'interface.

4.2 Compression/Décompression tar.gz

Pour cette seconde soutenance nous avons implémenter un second type permettant la compression de fichiers et dossiers, une extension de fichier bien connue des étudiants d'EPITA puisque tout nos skeleton de TP ou examens sont en ce format : le tar.gz.

L'implémentation de ce format de fichier a été possible grâce aux "dependancies" existantes en rust et qui ont été essentielles pour les imports de librairies que l'on a utilisé.

Ces libs nous ont permis de pouvoir facilement manipuler les fichiers d'un ordinateur via rust.

Pour pouvoir commencer a compresser, les librairies flate2 : :write : :GzEncoder, flate2 : :read : :GzDecoder et flate2 : :Compression nous ont permis de pouvoir accéder aux algorithmes ainsi que les utiliser pour compresser et décompresser les fichiers.

Bien sur, d'autres imports ont été utilisés comme la lib std : :fs : :File, std : :io, Path... afin de pouvoir correctement créer des fichiers, des dossiers trouver leur path et gérer les éventuelles erreurs.

```
use std::fs::File;
use std::io;
use std::path::Path;
use std::error::Error;
use flate2::write::GzEncoder;
use flate2::read::GzDecoder;
use flate2::Compression;
use tar::Builder;
use tar::Archive;
```

4.2.1 Compression

Pour compresser et décompresser en tar.gz nous avons créer 2 fonctions, une de compression et évidemment la seconde pour la décompression contrairement au ZIP cette fois ci aucune fonction auxiliaire n'a été nécessaire.

La fonction de compression prend deux paramètres sous forme de string en entrée, ces paramètres sont fournis par l'utilisateur directement et correspondent au nom du dossier qui va être créer et le fichier/dossier devant être compressé.

Dans un premier temps, grâce aux fonctions des libs précédentes le dossier va être construit, il est vide initialement.

Grâce aux libs flate et tar ce dossier va donc pouvoir être compressé, ces libs sont plus facilement manipulables il n'est pas nécessaire cette fois ci de déplacer tout les fichiers du dossier dans le nouveau puis apres de le compresser, on le fait directement grâce aux libs tar et flate.

```
pub fn tarcomp(source_dir: &str, target_file: &str) -> Result<(), Box<dyn Error>> {
```

Comme on peut le voir, la fonction de compression renvoie un `Result<(), Box<dyn Error>>` comme les fonctions ZIP.

Une fois le dossier créer, on applique les fonctions de compressions et à la fin la méthode "append dir all" permettant simplement l'ajout des fichiers dans l'archive.

```
tar.append_dir_all(".", source_dir)?;
tar.finish()?;
Ok(())
```

Cette fonction de compression est donc bien plus digestive visuellement que celle en zip puisque la majorité du travail se fait via les fonctions déjà disponibles via les libs.

4.2.2 Décompression

La décompression des fichiers tar a un déroulement extremement similaire à la compression, la fonction prend également 2 strings en paramètres, le dossier à décompresser et le dossier où stocker les fichiers présents dans le tar.gz.

On crée ensuite le dossier de stockage grâce à la lib std : :fs : :File, ensuite si dans la compression on utilise la fonction GZEncoder, ici on utilisera la fonction GZDecoder, on envoie le résultat dans une variable dont le contenu sera altéré par la fonction archive de la lib tar : :Archive.

Une fois cette étape finie il ne manque qu'à unpack le résultat pour terminer la décompression.

La partie compression/décompression de fichiers avance bien en accord avec les prédictions du cahier des charges.

Malgré les difficultés rencontrées avec la compression vidéo, maintenant que la compression tar.gz et image sont fonctionnels, il ne manque plus grand chose à faire pour la partie fonctionnelle du projet.

Pour la prochaine et dernière soutenance le projet sera donc fini, toutes les compressions/décompressions seront terminées et au vu du rythme de l'équipe la section bonus du cahier des charges pourra sûrement être réalisée. (Compression depuis le site web)

4.3 Compression Image/vidéo

4.3.1 Algorithmie

Pour La compression image, Je me suis inspire de "DEFLATE" un algorithme de compression de données sans perte notamment utilisés pour le format PNG et zip. Cet algorithme utilise deux algorithme différent, l'algorithme Lz77 et l'algorithme d'Huffman.

LZ77 est un algorithme de compression sans perte basé sur le principe de glissement de fenêtre. Il repose sur l'idée que les données contiennent souvent des séquences répétées. Plutôt que de stocker chaque occurrence répétée, l'algorithme remplace ces répétitions par des références à leur première apparition dans les données déjà analysées. Cela permet de réduire la taille totale de l'information, tout en conservant l'intégralité des données originales lors de la décompression.

Dans l'implémentation utilisée ici, une fenêtre de recherche glisse sur les données à compresser. À chaque position, l'algorithme examine la fenêtre pour détecter la plus longue correspondance possible entre la portion actuelle des données et une séquence précédente. Si une correspondance d'au moins trois octets est trouvée, elle est représentée par un token (Type1) contenant deux informations : l'offset (distance entre la position actuelle et le début de la séquence répétée) et la longueur de la séquence. Si aucune correspondance suffisante n'est trouvée, l'algorithme enregistre le byte actuel sous forme de token (Type2).

Cette approche permet de produire un flux compressé composé uniquement de ces deux types de tokens. Chaque jeton contient l'information minimale nécessaire à la reconstruction exacte des données d'origine. La phase de décompression lit les tokens les uns après les autres : si un type2 est rencontré, il est ajouté tel quel à la sortie ; si un type1 est trouvée, les octets correspondants sont copiés depuis les données déjà décompressées.

Ce système est simple et efficace, particulièrement adapté aux contenus où des motifs reviennent fréquemment comme dans les images. Dans ce projet, LZ77 est utilisé pour compresser chaque image ou les différences entre deux images. Cela permet de réduire considérablement la quantité de données à sauvegarder ou à transmettre.

L'algorithme de Huffman est une méthode de compression de données sans perte basée sur la fréquence des symboles. Il permet de réduire la taille d'un fichier en attribuant des codes binaires plus courts aux symboles les plus fréquents, et des codes plus longs aux symboles rares.

Huffman est utilisé en complément d'un premier niveau de compression basé sur l'algorithme LZ77. Une fois les données (soit les frames complètes, soit les différences entre deux frames) compressées avec LZ77, on applique l'encodage Huffman pour compresser encore davantage. L'implémentation commence par une analyse des données pour déterminer la fréquence d'apparition de chaque octet. Ces fréquences sont ensuite utilisées pour construire un arbre de Huffman, un arbre binaire dans laquelle chaque octet est représenté par une feuille. Les chemins dans l'arbre définissent les codes binaires attribués à chaque symbole : plus un symbole est fréquent, plus son code est court.

Une fois l'arbre construit, les données sont parcourues, et chaque symbole est remplacé par son code binaire. Cela produit une longue séquence de bits, qui est ensuite transformée en octets pour faciliter le stockage. Au moment de la décompression, cette séquence est retransformée en bits, et l'arbre de Huffman est utilisé pour retrouver les symboles d'origine en suivant les chemins de bits dans l'arbre.

Dans cette version du projet, un arbre de Huffman générique est utilisé à la décompression, ce qui simplifie l'implémentation mais limite l'efficacité et la fiabilité du décodage. Une amélioration envisageable serait de transmettre également l'arbre utilisé lors de l'encodage, afin d'assurer une reconstruction fidèle des données.

Ces deux algorithmes permettent de compresser de manière efficace une image. Compresser une vidéo revient à compresser un tas d'images à la suite. Par conséquent, nous utiliserons ce même algorithme pour la compression vidéo. Il suffit de découper notre vidéos en frames de compresser ces frames et de recomposer la vidéo à partir de ces frames. Cependant, des complications sont apparues.

4.3.2 Problèmes rencontrés

Après avoir réalisé ces deux algorithmes, je me suis heurté à un problème majeur pour la compression vidéo, mon code était trop lent, pour une vidéo de 3 secondes il me fallait 30 minutes pour que mon code finisse de compiler. Afin de réduire ce temps, j'ai dû m'intéresser au Group of pictures(GOP). La compression vidéo est faite de la façon suivante : chaque frame de la vidéo est découpé en image et les algorithmes sont appliqués sur chacune de ces images. Ensuite il suffit de les remettre les unes après les autres pour recréer la vidéo.

La méthode GOP consiste à regrouper les images en groupe (des images se suivant) car partant du principe que les images successives d'une vidéo sont sensiblement similaires. La première image est key frame c'est à dire que nous allons lui appliquer nos algorithmes entièrement. Les autres sont appelés delta frames, elles

ne contiennent que les différences avec la key frame ce qui permet d'être plus rapide lors de l'exécution des algorithmes. j'ai également utilisé rayon, une bibliothèque permettant de faire des tâches en parallèle. Chaque GOP étant indépendant ils peuvent et sont fait en même temps. Ce qui permet de réduire le temps de compilation à 60 secondes pour une vidéo de 3 secondes.

Une des améliorations envisagées serait notamment de réaliser plus de tâches en parallèle grâce à rayon.

4.4 Site Web

Pour la seconde soutenance , il y a eu pas mal d'améliorations qui vont dans le sens de ce qui était prévu jusque là. En bref résumé avant de rentrer dans les détails, ces derniers consistent en la possibilité de télécharger le projet et le rapport directement depuis le site, ainsi qu'un léger travail sur l'esthétique et quelques ajouts pour le confort de l'internaute.

4.4.1 Améliorations graphiques

Cette version du site devant être beaucoup plus aboutie, nous nous devions d'améliorer l'esthétique générale du site, et cela passe par une page d'accueil plus attrayante et visuellement parlante.

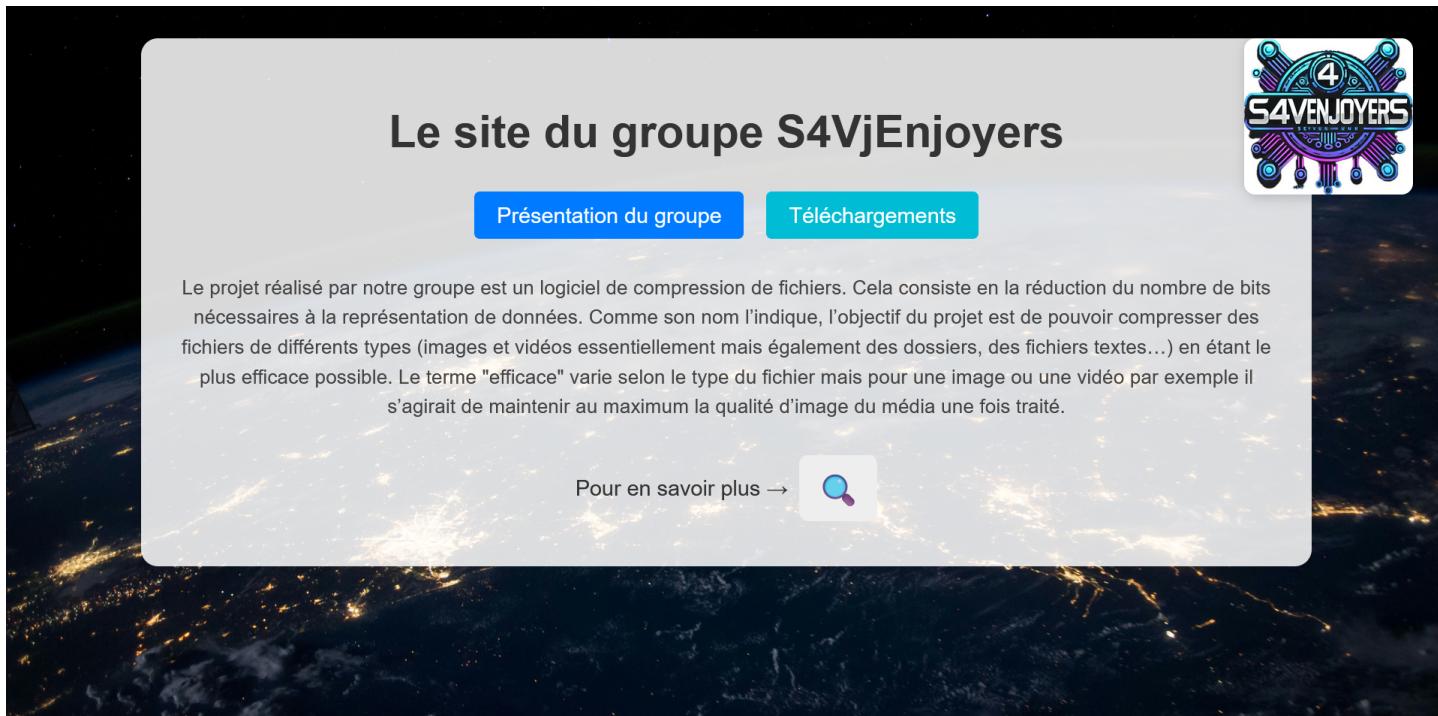


FIGURE 1 – page d'accueil du site

Comme on peut le voir sur la page d'accueil , il y a eu l'ajout d'un fond qui remplace donc le fond blanc monotone et basique, bien que nous n'ayont pas ici affaire au top niveau concernant les possibles fonds de page, je pense que cela est correct car doté de peu de détails, juste ce qu'il faut.

Je rappelle brièvement que tout le site est codé à la main, en utilisant HTML et CSS et sans aucune librairie utilisée (au cas où vous aimiez tellement notre site que vous en veniez à vous poser des questions). Le site se comporte donc d'une page d'accueil, dotée d'une partie explication du projet, ainsi que d'un bouton

Présentation du groupe

Voici le site internet de notre groupe de projet **S4VjEnjouers**, composé des membres suivants :

Timéo Berthault
Lucas Peres
Aboubakar Charf

Le site comprendra de nombreuses améliorations au fur et à mesure que le semestre progresse.

Les héros du groupe



[Retour à l'accueil](#)

FIGURE 2 – page de présentation du groupe

pour en savoir plus sur le sujet, qui redirige vers la page wikipedia qui concerne la compression de données (vous l'aurez compris, c'est le sujet de notre projet). Sur ce même accueil il s'y trouve un bouton téléchargement qui nous amène comme indiqué, sur la page liée au téléchargement du projet ainsi que du rapport, et un bouton nous amenant à la page de présentation du groupe, avec nos photos et nos noms. On peut notamment retrouver le logo de notre groupe en haut à droite de la page d'accueil, que nous affichons avec fierté.

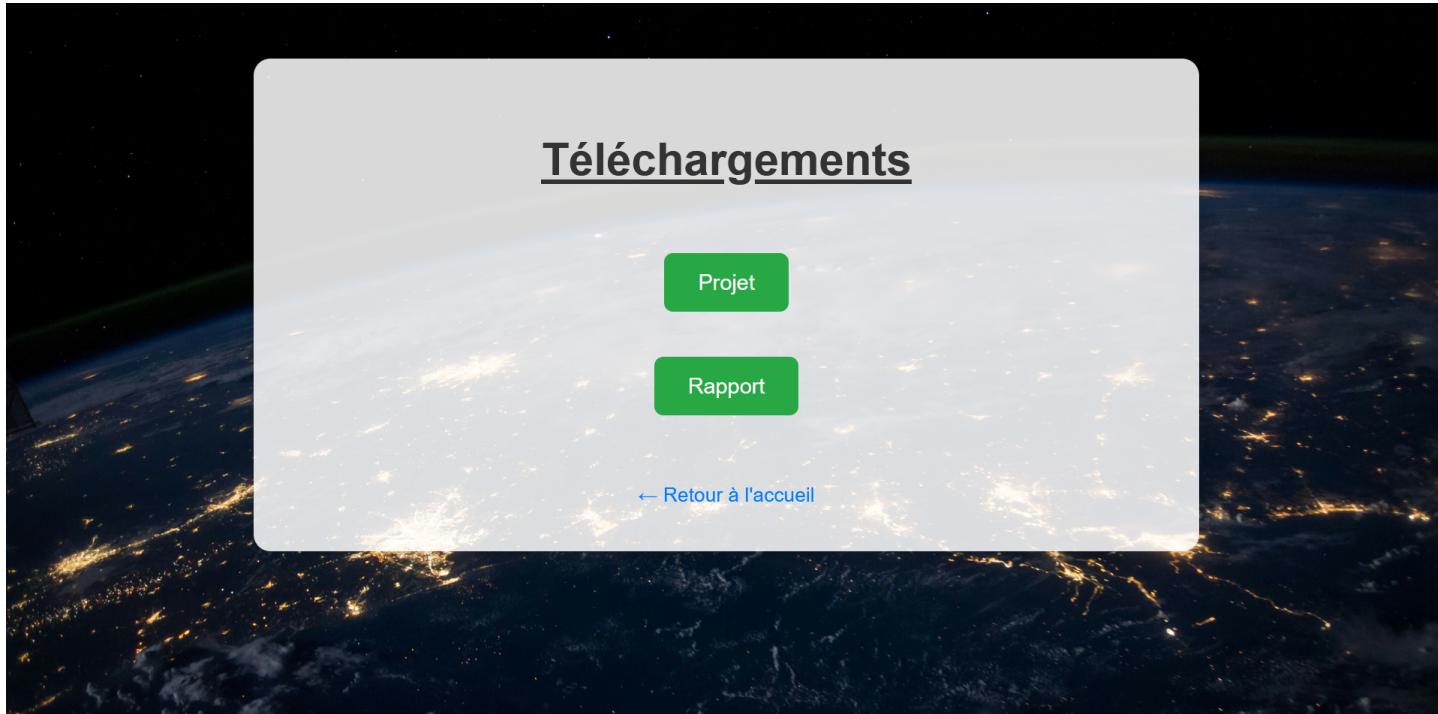


FIGURE 3 – page de téléchargement du projet et du rapport

5 Conclusion

Pour conclure, le projet avance plus qu'efficacement et tout ce qui était requis pour les premières et deuxième soutenances ont été accomplis.

Pour la deuxième soutenance, Nous avons amélioré l'interface afin de la relier aux fonctions déjà fonctionnelles et d'en améliorer l'esthétique. Les différentes compressions et décompressions ont également été étoffées avec l'ajout de nouveaux algorithmes et d'extensions supportées. Le site web, quant à lui, a fait l'objet d'une amélioration constante et n'a en aucun cas été négligé comparé aux autres parties du projet.

Concernant la troisième et dernière soutenance : Nous nous consacrerons à l'amélioration des fonctionnalités déjà présentes, ainsi qu'à la compression vidéo nécessitant de lourdes améliorations.

Parmi celles-ci :

- Error handling (Gestion d'erreurs) de l'interface graphique ;
- Completer le READ ME avec un guide d'utilisation ;

- Améliorer les fonctionnalités/optimisations des compressions

La répartition sera modifiée pour que Aboubakar, après avoir complété le site final, puisse venir en aide aux parties en retard ou qui ont besoin d'aide, notamment pour la compression vidéo.

6 Sources

Les documentations utilisees pour notre projet sont :

Pour l'interface graphique :

- <https://docs.rs/gtk/latest/gtk/>

Pour les differentes compressions :

- <https://docs.rs/image/latest/image/>
- <https://doc.rust-lang.org/std/path/struct.Path.html>
- <https://doc.rust-lang.org/std/iter/struct.Zip.html>

Les sources sont principalement :

- <https://github.com/>
- <https://www.youtube.com/>
- <https://fr.wikipedia.org/wiki/Wikipedia>
- <https://chat.openai.com/> (pour la creation d'image)