

+



PLATEFORME DE SIGNATURE D'IMAGES BASE SUR LA STEGANOGRAPHIE

ABSTRACT

La stéganographie est l'art de dissimuler des informations au sein d'un media tel qu'une image ou plus généralement un fichier. Dans ce document sera résumé une vue ainsi qu'une description d'une implémentation de celle-ci au sein d'une application web.

Par Ivo COSTA CUNHA

Université Côte d'Azur – Master MIAGE INTENSE

Table des matières

1. Le cadrage du projet.....	2
2. Recueil d'exigences.....	3
3. Etat de l'art	4
3.1. Le sujet	4
3.2. Les exigences en matière de conception d'un système stéganographique	5
3.3. Les effets de la compression sur une image	7
3.4. L'état de l'art de la stéganographie.....	23
3.5. Evaluation des différentes techniques	29
3.6. Choix de l'implémentation finale	30
4. Réalisation.....	31
4.1. Technologies utilisées	31
4.2. Logiciels utilisés	32
4.3. Outils de gestion de projet.....	33
4.4. Architecture	34
4.5. Le client frontend	36
4.6. L'API backend	40
5. Conclusion.....	45
5.1. Tâches effectuées.....	45
5.2. Perceptives futures	46
5.3. Difficultés rencontrées.....	47
5.4. Bilan personnel	47

1. Le cadrage du projet



Ivo Costa Cunha

Etudiant en master MIAGE INTENSE

Responsable : Annie BLANDEL

Encadrant : Gregory Galli

Le projet a donc été développé par moi, ci-dessus.

On était deux étudiant en début de projet, Abdelrazak EL GHZZAZ et moi, cela dit celui-ci a quitté ce projet vers le début. Il m'a beaucoup aidé sur la recherche néanmoins.

On était et maintenant je suis encadré par Gregory Galli, un de mes professeurs de Master dans les matières de base de données et développement logiciel en général, dont je remercie l'aide qu'il a pu m'apporter au cours du projet ainsi que la direction qu'il m'a donnée.

La direction m'a permis d'avoir des objectifs prioritaires et de définir un MVP, Minimum Viable Product, et j'en ai définitivement besoin car je change d'objectifs vite.

Je remercie également l'organisation de la MIAGE en général qui m'a donné la possibilité de travailler sur ce projet qui s'est avéré très intéressant que ce soit du côté technique, via les technologies utilisées, que du côté du but attendu, ainsi que son intérêt dans le futur étant donné que celui-ci s'intègre bien dans les problématiques qui se posent aujourd'hui et dans le futur.

Le code et autres scripts seront contenus dans le repository GitHub suivant :

<https://github.com/IvoCostaCunha/stegano-image>

2. Recueil d'exigences

Comme tout projet des exigences ont été stipulés avant son début.

Ce recueil m'a permis de définir un MVP a priorisé, les éléments du MVP étant en vert et en gras.

Etat de l'art
<ul style="list-style-type: none">• Produire une analyser complète des différentes techniques et implémentations existantes de la stéganographie qui offrent une résistance importante à la compression.• Choisir avec justification une des implémentations analysées auparavant.
Construction de la plateforme de signatures d'images
<ul style="list-style-type: none">• Mettre en place un système d'inscription sécurisé.• Mettre en place un système d'identification sécurisé.• Mettre en place une fonction d'upload d'images qui seront automatiquement signés pour les utilisateurs connectés.• La signature d'images doit être résistante à la compression.• La signature d'images doit altérer le moins possible la qualité de l'image originale.• Mettre en place une page ou l'utilisateur peut upload une image et obtenir les informations de l'auteur de l'image via une vérification de la signature.• L'application doit tenir compte des meilleures pratiques en matière de sécurité en utiliser des mécanismes d'authentification robustes.• La confidentialité des données utilisateurs doit être assuré.• Le front est libre au niveau de l'implémentation.• Le backend doit être en Flask, un Framework Python.

3. Etat de l'art

3.1. Le sujet

3.1.1. Contexte

Avec la multiplication des images numériques et l'omniprésence d'Internet, la facilité avec laquelle les images peuvent être dupliquées, modifiées et diffusées, un problème de droit d'auteur a été soulevé.

Ignorant les débats philosophiques sur la libre circulation des images, la signature numérique des images apparaît comme une solution permettant de garantir l'intégrité, la propriété intellectuelle et l'authentification du contenu, en incorporant celle-ci au sein même des fichiers de façon caché.

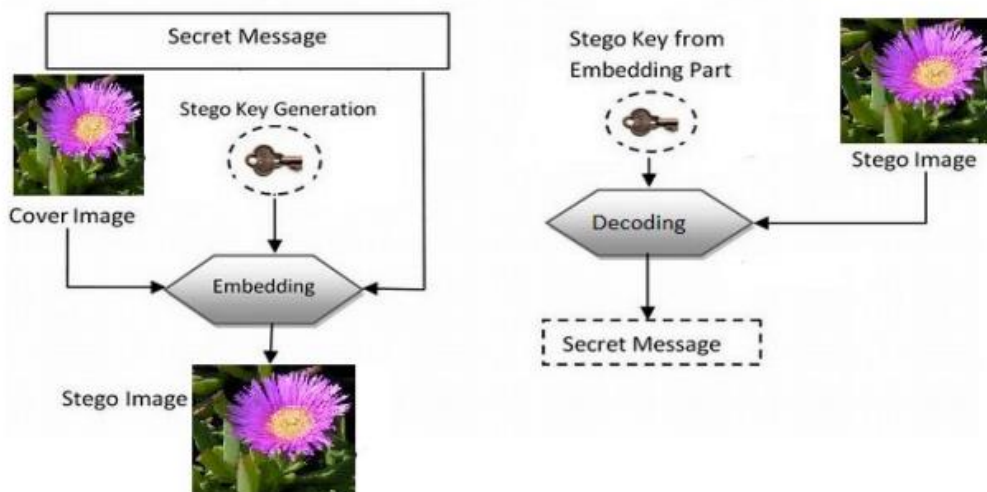
Dans ce contexte une solution sera présente dans ce document avec une veille faite avant que cette implémentation ait été faire.

3.1.2. Définition

La stéganographie permet de cacher des données de type ASCII¹ par exemple au sein d'un fichier, de façon à ce que leur présence soit imperceptible aux regards extérieurs. Son objectif est de dissimuler les données de telle manière à que les données cachées ne soient pas facilement visibles, et avec la cryptographie permet qu'aucune personne autre que l'expéditeur et le destinataire ne puissent les décrypter.

La cryptographie est utilisée donc en complémentarité de la stéganographie.

Celle-ci vise à sécuriser les communications en transformant les données en une forme incompréhensible pour les personnes non autorisées. Elle utilise des techniques de chiffrement pour protéger le contenu, mais l'existence même de la communication n'est pas dissimulée. Cette étape est typique faite avec l'ensemble clé publique privée.



¹ <https://www.ascii-code.com/fr>

Figure 1 – Les étapes de transfert d'une image non signé à une image signé envoyé

Donc, la stéganographie se concentre sur la dissimulation des données, tandis que la cryptographie se concentre sur leur sécurisation. Les deux techniques sont combinées pour renforcer la confidentialité des communications en cachant les données sensibles à l'aide de la stéganographie et en les protégeant par le chiffrement cryptographique.

3.2. Les exigences en matière de conception d'un système stéganographique

Pour concevoir un système de signature d'images efficace, il est important de définir les exigences et les caractéristiques clés. La figure ci-dessous présente un aperçu des principales exigences des techniques de signature. Bien que cette figure s'adresse à la technique de Watermark, les exigences peuvent s'appliquer à l'implémentation des techniques de stéganographie au sein des données d'un fichier.

Ces exigences permettent d'en mesurer les performances.

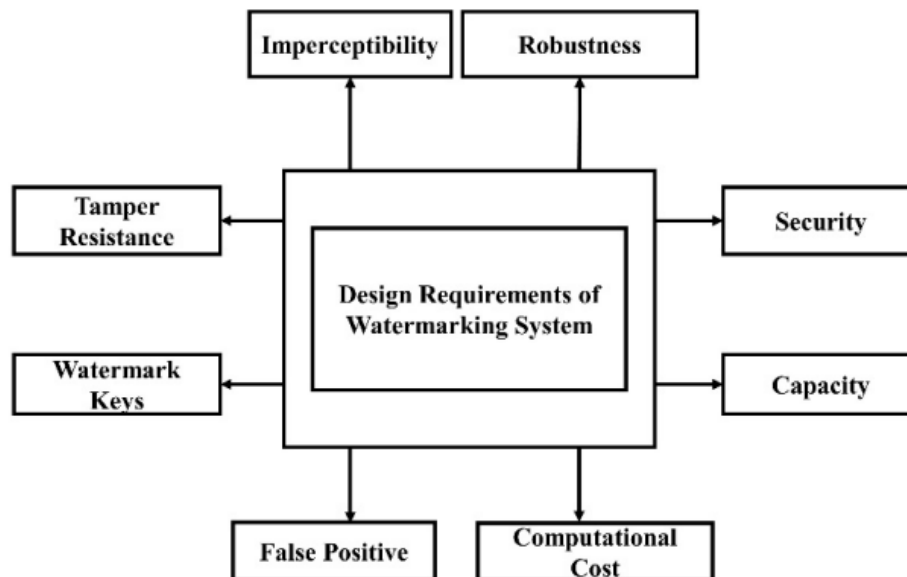


Figure 2 – Les exigences d'un système de Watermark

Imperceptibilité

Les données cachées ne doivent pas être facilement reconnues à l'œil ou par des outils de recherche. Il faut écrire les données de façon erratique et non linéaire.

Robustesse

Aujourd'hui la plupart des sites effectuent une compression des images uploadées/échangées. Cela est principalement dû à deux raisons : l'optimisation des performances et la gestion de l'espace de stockage.

Comme tous les fichiers d'images sont compressés il est important que l'implémentation survive à des cycles de compressions et décompression du moins dans les cas prévus, car il est très difficile de faire survivre des données à des compression destructives.

Sécurité

La clé stéganographique doit permettre d'identifier son propriétaire de façon sécurisé, et le lien entre ces 2 acteurs doit lui aussi être sécurisé.

Capacité

Toute image doit pouvoir être signé.

Résistance à la falsification

Il est important qu'une clé ne puisse pas être implémenté par un tier non autorisé. La création de cette clé doit donc suivre des algorithmes sécurisés à cette fin.

Coût computationnel

Dans une moindre mesure les coûts au niveau des serveur qui permettent l'implémentation doivent être pris en compte et réduits autant que possible.

Faux positif

Le système de reconnaissance des clés ne doit pas associer des clés a de mauvais propriétaires et vis-versa.

La clé stéganographique implémentée (dans ce cette implémentation)

Il est nécessaire de trouver un équilibre entre l'implémentation de la clé stéganographique au sein du fichier et la qualité visuelle de celui-ci afin de garantir que les données intégrées dans celui-ci détériorent le moins la qualité visuelle.

De plus il est nécessaire que la clé puisse être écrite dans des fichiers de petite taille, il faut donc trouver un juste milieu entre sécurité et l'utilisabilité également.

3.3. Les effets de la compression sur une image

La compression est le fait de réduire le poids d'une image en réduisant la taille du fichier qui la décrit. Plusieurs techniques de compression existent, certaines plus efficaces que d'autres dans la réduction de la taille, et certaines plus ou moins dites destructives que d'autres. Généralement plus la technique est destructive plus l'image perd en taille mais aussi en qualité et fidélité par rapport à l'originale.

Dans ce contexte la destructivité comprend aussi les données insérées dans le fichier car oui les données seront insérées au sein de l'encodage des pixels. Si un pixel est substitué, la partie des données insérées dans le document que comprend ce pixel l'est aussi ce qui corrompt ces données cas de compression et les rend donc illisibles. Cela casse plusieurs exigences vues précédemment.

C'est pourquoi une étude des méthodes de compression les plus connues a été faite.

A titre d'exemple, le format jpeg change directement la façon dont une image est encodée de rgb à Y'CbCr. Si l'on cache notre message sur chaque pixel codé en rgb alors notre message sera perdu lors d'une compression jpeg.

Certaines méthodes de compression sont plus utilisées que d'autres. Sur le web on utilise souvent du jpg ou du png et donc ces 2 méthodes seront étudiées ici.

3.3.1. Le format jpg²

Le format Y'CrCb

Pour commencer le format jpg code un pixel non dans le format classique RGB (Rouge Vert Bleu) mais sur un format Y'CrCb.

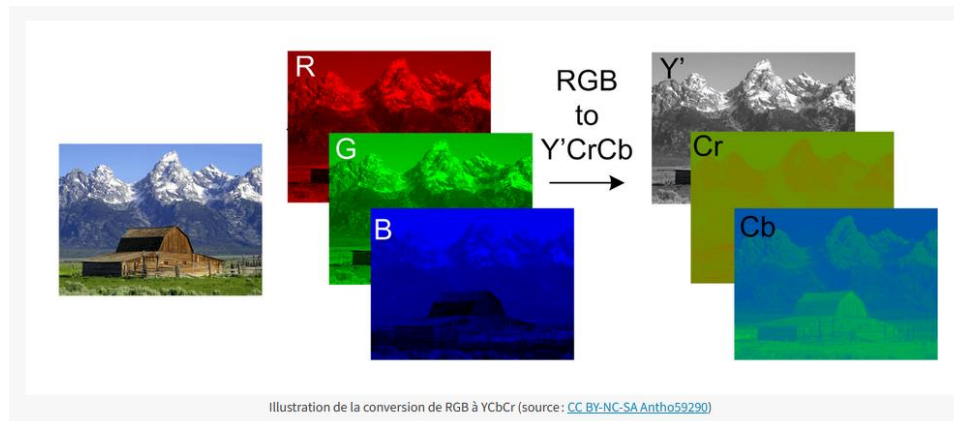


Figure 3 - Schéma RGB vers Y'CrCb

Ce format a plusieurs avantages en termes de compression :

- L'œil humain voit mieux les contours que l'intensité des couleurs.
- Les contours ont tendance à être chaotiques dans les images mais pas les couleurs.

Par conséquent, en réduisant la précision des couleurs sans toucher les contours, on peut alors réduire la taille d'une image sans modifier de façon significative les formes.

Chroma subsampling

Le chroma subsampling est une technique *pas forcément utilisée* avec laquelle on réduit la taille des couches Cr et Cb.

La notation J:a:b est la notation courante en termes de chroma subsampling et elle définit le niveau de downsampling.

	4:4:4	4:2:2	4:2:0 (le plus répandu)																								
Pixels d'intensité	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>5</td><td>6</td><td>7</td><td>8</td></tr></table>	1	2	3	4	5	6	7	8	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>5</td><td>6</td><td>7</td><td>8</td></tr></table>	1	2	3	4	5	6	7	8	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>5</td><td>6</td><td>7</td><td>8</td></tr></table>	1	2	3	4	5	6	7	8
1	2	3	4																								
5	6	7	8																								
1	2	3	4																								
5	6	7	8																								
1	2	3	4																								
5	6	7	8																								
Pixels de bleu	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>5</td><td>6</td><td>7</td><td>8</td></tr></table>	1	2	3	4	5	6	7	8	<table><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4	<table><tr><td>1</td><td>2</td></tr></table>	1	2										
1	2	3	4																								
5	6	7	8																								
1	2																										
3	4																										
1	2																										
Pixels de rouge	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>5</td><td>6</td><td>7</td><td>8</td></tr></table>	1	2	3	4	5	6	7	8	<table><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4	<table><tr><td>1</td><td>2</td></tr></table>	1	2										
1	2	3	4																								
5	6	7	8																								
1	2																										
3	4																										
1	2																										

Les différents réglages du chroma subsampling avec JPEG.

Figure 4 - Les différents paramètres du chroma subsampling

² <http://compression.fiches-horaires.net/la-compression-avec-perte-1/le-compression-jpeg/>

Le premier chiffre est le nombre de pixels d'intensité conservés pour 4 pixels à l'écran sur

Chaque ligne.

Le second chiffre est le nombre de pixels de couleur conservés pour 4 pixels à l'écran sur des lignes paires.

Le troisième chiffre est le nombre de pixels de couleurs affichés pour 4 pixels à l'écran sur les lignes impaires.

A noter que cette technique est très grossière et a pour effet de faire des tâches de couleurs sur une image et est donc rarement utilisée, hormis dans les compressions les plus importantes. Gimp utilise par défaut du 4:4:4 ce qui revient à ne pas utiliser cette technique.

Le découpage en blocs

Un des points importants du jpeg est que l'on travaille sur des blocs de 8x8 pixels donc dans les faits le chroma downsampling est fait sur chaque bloc et pas l'image en entier, et cela sur chaque couche.

Donc si un des axes de l'image n'est pas un multiple de 8x8 alors des artefacts de compression peuvent apparaître lors de compressions importantes.

On appelle ces blocs des MCU, aka Minimum Coded Units.

Le codage DCT

DCT signifie Discrete Cosine Transform et est une fonction qui permet dans notre cas de transformer le tableau des MCU. Le tableau des MCU devient alors un tableau avec des rayures plus ou moins denses et plus ou moins rapprochées, comme sur l'exemple qui suit.

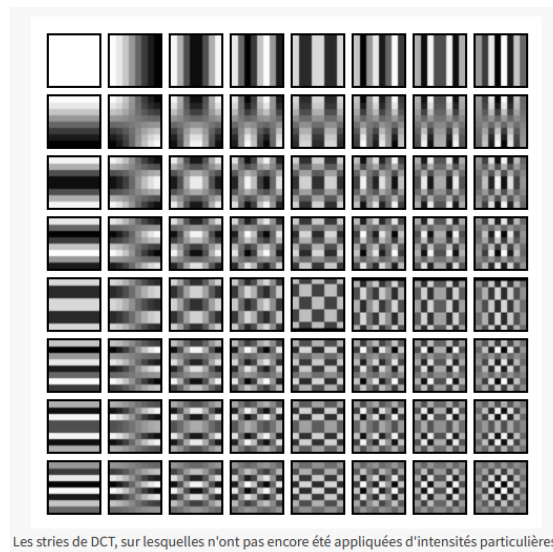


Figure 5 - Visuel de l'encodage DCT

La fonction DCT permet de déterminer où sont les détails de l'image et où elle est moins détaillée. En déterminant où l'image est plus détaillée on peut fortement réduire la qualité des blocs plus détaillés sans pour autant toucher à la fidélité de l'image en général, en termes de contraste et/ou illumination.

On obtient par exemple à partir du visuel ci-dessus la matrice suivante.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 6 - Matrice exemple DCT (ensemble [0;255])

La matrice DCT est de base sur l'ensemble [0;255], mais est transposée sur [-128;127].

$$g = \begin{matrix} & \begin{matrix} x \\ \longrightarrow \end{matrix} \\ \begin{matrix} \downarrow y. \end{matrix} & \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \end{matrix}$$

Figure 7 - Matrice exemple DCT (ensemble [-128;127])

L'étape suivante est d'appliquer la formule ci-dessous à la matrice ci-dessus.

$$G_{u,v} = \frac{1}{4} \alpha(u) \alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 g_{x,y} \cos \left[\frac{(2x+1)u\pi}{16} \right] \cos \left[\frac{(2y+1)v\pi}{16} \right]$$

where

- u is the horizontal [spatial frequency](#), for the integers $0 \leq u < 8$.
- v is the vertical spatial frequency, for the integers $0 \leq v < 8$.
- $\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } u = 0 \\ 1, & \text{otherwise} \end{cases}$ is a normalizing scale factor to make the transformation [orthonormal](#)
- $g_{x,y}$ is the pixel value at coordinates (x, y)
- $G_{u,v}$ is the DCT coefficient at coordinates (u, v) .

Avec cette formule on peut obtenir la matrice DCT en 2 dimensions, et une fois celle-ci appliquée à la matrice DCT précédente (g) on obtient à la matrice ci-dessous (G).

$$G = \begin{matrix} & & & u & & & & \\ & & & \longrightarrow & & & & \\ \begin{matrix} \downarrow v \\ \end{matrix} & \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.12 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.87 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \end{matrix}$$

Figure 8 - Matrice exemple DCT (2d)

Dans cette matrice le premier coefficient en haut à gauche est appelé le coefficient **DC (Constant Component)** et tous les autres **AC (Alternative Components)**.

Le DC définit la teinte du block.

La quantification

La quantification permet de déterminer par combien on divisera la qualité de chaque bloc DCT. L'idée est de diviser par un nombre plus grand les blocs les plus détaillés et par des nombres plus petits ce qui sont le moins détaillés.

Le but de la matrice de quantification DCT est d'atténuer les grandes fréquences de la matrice DCT et de ramener beaucoup de coefficients de celle-ci à 0 ce qui simplifie sa compression.

Pour cela on utilise une matrice de quantification qui va donc faire correspondre un nombre à chaque block. Ci-dessus on peut voir une matrice de quantification standard.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 9 - Exemple de matrice de quantification standard utilisé

On divise ensuite la matrice G précédente par cette matrice du standard jpg³ et on arrondi à l'entier le plus proche comme le fait la formule suivante.

$$B_{j,k} = \text{round} \left(\frac{G_{j,k}}{Q_{j,k}} \right) \text{ for } j = 0, 1, 2, \dots, 7; k = 0, 1, 2, \dots, 7$$

where G is the unquantized DCT coefficients; Q is the quantization matrix above; and B is the quantized DCT coefficients.

Cette formule est donnée par le standard JPEG⁴ mais des logiciels comme Photoshop de Adobe utilisent leurs propres formules.

Cette matrice s'applique pour une qualité de 50% mais il est possible de réduire encore plus la qualité.

³ <https://en.wikipedia.org/wiki/JPEG#Quantization>

⁴ <https://jpeg.org/jpeg/>

On obtient ensuite la matrice quantifié DCT qui correspond dans l'exemple à la suivante.

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Figure 10 - Matrice quantifié DCT

Le codage entropique

Lorsque l'on veut écrire dans le fichier final, on a besoin de 2 choses, les données compressées, et les tables de Huffman qui permettent de les décompresser.

Les tables de Huffman sont la matrice de quantification mais réduite en taille par l'encodage entropique suivi par l'algorithme de Huffman.

L'encodage entropique permet d'encoder donc la matrice de quantification mais cela en zigzag comme on peut le voir sur le schéma suivant.

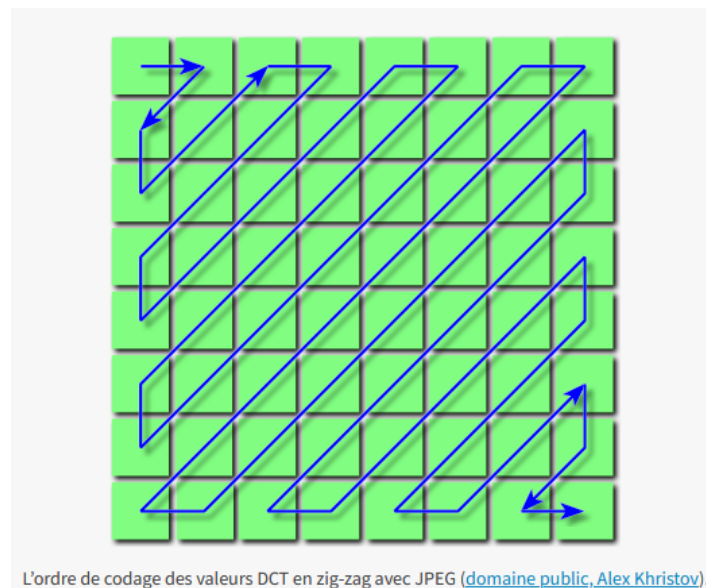


Figure 11 - Ordre d'encodage des valeurs DCT en zigzag

On obtient donc les coefficients suivants :

```
-26
-3  0
-3 -2 -6
 2 -4  1 -3
 1  1  5  1  2
-1  1 -1  2  0  0
 0  0  0 -1 -1  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0
 0  0  0  0
 0  0  0
 0  0
0
```

Figure 12 - Application du codage entropique

Finalement il est appliqué un algorithme Deflate⁵ qui s'apparente à Huffman mais est modifié partiellement par rapport à du Huffman classique.

Le lien en bas de page détaille cet algorithme.

⁵ <https://compression.fiches-horaires.net/la-compression-sans-perte/deflate-lalgorithme-que-vous-retrouvez-partout/#2-le-codage-huffman-applique-a-deflate>

Récapitulatif

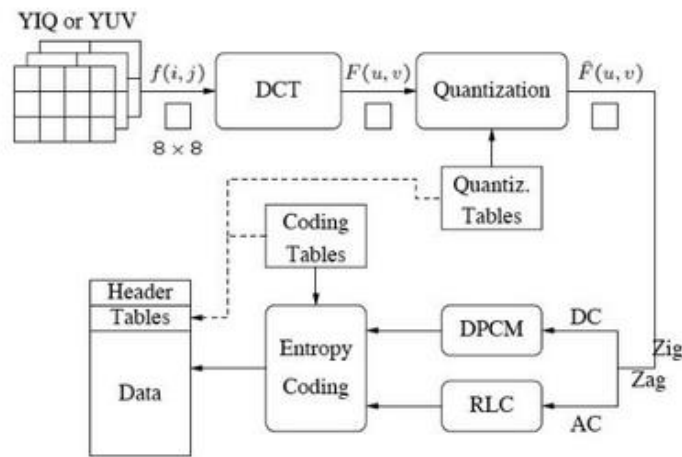


Figure 13 - Schéma récapitulatif JPG

Cela étant dit, cette approche est appelée **l'encodage séquentiel** JPG, mais existe aussi **l'encodage progressif**.

Là où l'encodage séquentiel travaille block par block l'encodage progressif lui travaille sur plusieurs blocs en mêmes temps regroupés dans des "scans" là où l'encodage séquentiel aura un seul "scan".

Le type d'encodage est indiqué dans l'entête du fichier.

Cela est l'encodage du fichier JPG compressé mais pour le lire il faut le décoder pour cela les mêmes étapes sont faites mais dans l'autre sens avec des formules inversées.

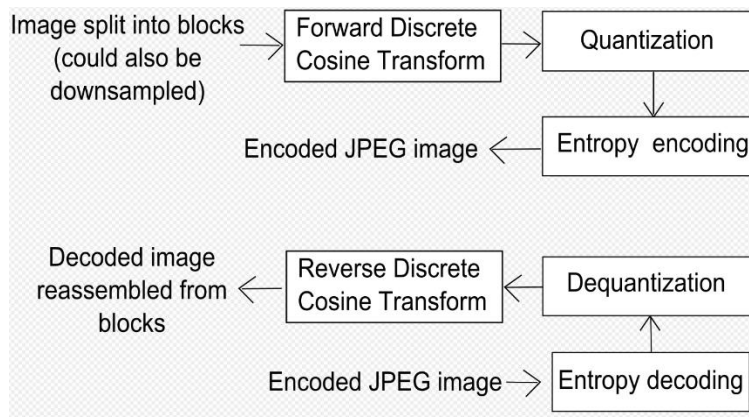


Figure 14 - Cycle compression/décompression JPG

Les segments

Une fois encodé le fichier devient du binaire mais alors comment faire pour en reconnaître les différentes parties aussi appelées segments⁶. Pour cela on utilise des balises.

Ces balises sont des marqueurs qui indiquent ce qui suit.

Ils sont en 4 étapes :

[1 octet] L'indicateur de début de marqueur, toujours l'octet 0xff

[1 octet] Le type de marqueur, un octet entre 0x00 et 0xfe

[2 octets] (Optionnel) La taille du segment (sur 2 octets)

[variable] Les données du segment en question

Le fichier commence toujours par un marqueur SOI (Start of Image), codé ffd8 sans plus d'informations et se finit par un marqueur EOI (End of Image), codé ffd9 toujours sans plus d'informations.

```
00000000  FF D8 FF E1 00 94 45 78 69 66 00 00 4D 4D 00 2A  . . . . . E x i f . . M M . *
00001D30  D2 81 50 72 51 FA 5F FF D9  . . . . . P r Q . _ . .
```

Figure 15 - Exemple SOI & EOI

Ici on a un début et fin de .jpg à titre d'exemple.

Il existe beaucoup d'autres segments, en plus de segments liés à des logiciels comme Adobe Photoshop mais seuls les plus importants seront présentés ici.

Le segment **SOF0 ou Start of Frame (0)** indiquent les propriétés importantes de l'image, hauteur, largeur, nombre de composants et la valeur subsampling. Cela commence par **FFC0**.

Le segment **SOF2** présente les mêmes informations mais indique que ce fichier jpeg a été codé avec des DCT progressifs. Il commence par **FFC2**.

Le segment **DHT ou Define Huffman Tables** contient les tables de Huffman et commence par **FFC4**.

Le segment **SOS aka Start of Scan** indique le début du scan de l'image et est le début est **FFDA**. Dans le cas d'un encodage séquentiel il n'y aura qu'un seul scan mais en cas d'encodage progressif il y en aura plusieurs et dans chaque cas ils sont suivis de données codées de façon entropique.

Voir https://en.wikipedia.org/wiki/JPEG#Syntax_And_Structure pour les autres segments de base.

⁶ <https://www.disktuna.com/list-of-jpeg-markers/>

3.3.2. Le format PNG

Le format **PNG (Portable Network Graphics)** est un format de compression sans perte et qui supporte la transparence via les alpha channel contrairement à JPG.

Le format PNG est utilisé dans les pages web mais aussi pour tout ce qui est édition d'image car il permet de sauvegarder facilement des images sur plusieurs phases d'édition dans pertes dans la qualité de l'image.⁷

La profondeur de la couleur⁸

Il existe plusieurs types d'images PNG dépendantes de la "profondeur de la couleur".

La "profondeur de la couleur" correspond à quelles parties d'un pixel sont sauvegardés.

Typiquement un pixel est composé des 3 octets :

- 1 pour la valeur du rouge sur 256 valeurs stockées dans un octet
- 1 pour la valeur du vert sur 256 valeurs stockées dans un octet
- 1 pour la valeur du bleu sur 256 valeurs stockées dans un octet

Donc par exemple sur une image de 200x200 pixels on aura :

$3 \text{ octets} * 200 * 200 = 120\,000 \text{ octets} (0,12 \text{ Mo})$

Comme un octet est composé de 8 bits alors on parle d'une image PNG avec une profondeur de 24 bits.

Il existe d'autres formats de profondeur de couleur de type PNG.

Le **format 32bits** fonctionne comme précédemment mais pour chaque pixel on ajoute la chaîne de transparence aussi codé sur 1 octet et donc comme $4 * 8 = 32$, on parle d'un png 32bits. Dans ce cas on aurait alors une image d'une taille de $4 * 200 * 200 = 0.16 \text{ Mo}$.

Il existe aussi le **format 8bits**, celui-ci fonctionne différemment des précédents.

Ici au lieu d'utiliser un format RGB(T) pour chaque pixel on utilise une palette codée sur 256 valeurs donc pour chaque pixel. **Chaque pixel sera codé sur seulement 1 octet**, donc un format PNG 8 bits.

⁷ <http://www.libpng.org/pub/png/book/part1.html>

⁸ <https://compress-or-die.com/Understanding-PNG>

Suite à des tests sur des images compressés en jpg j'ai eu les résultats suivants :

Qualité %	% de pixels égaux
95	0.65333
90	0.6
80	0.46667
70	0.58667
60	0.49333
50	0.52

Le script Python pour obtenir ces valeurs est dans le répertoire « others/pixel_variance/ » du repository GitHub.

On voit que les pertes ne sont pas linéaires et que même à 95% de qualités 35% des pixels sont perdus, ce qui ne permet pas d'écrire sur les pixels de manière sûre.

La compression PNG

La compression PNG est composée de 3 étapes.

- Préfiltration (pre-filtering)
- L'encodage basé sur les dictionnaires (LZ77⁹)
- L'encodage entropique via l'algorithme de Huffman (vue dans la section JPEG)

Le tableau ci-dessus représente les valeurs par défaut à titre d'exemple.

1	2	3
1	2	3
1	2	3

Les 5 filtres sont¹⁰ :

1. Aucun

Comme son nom l'indique, aucun filtre n'est appliqué.

1	2	3
1	2	3
1	2	3

⁹ https://en.wikipedia.org/wiki/LZ77_and_LZ78

¹⁰ <https://www.w3.org/TR/PNG-Filters.html>

2. Sub

Garde la différence avec la valeur précédente en ligne de droite à gauche.

1	1	1
1	1	1
1	1	1

3. Up

Garder la différence avec la valeur précédente en colonnes de bas à en haut.

1	2	3
0	0	0
0	0	0

4. Average

Prédiction d'un pixel avec la moyenne de celui en haut et celui à gauche.

1	2	3
1	3	2.5
1	3	2.5

5. Paeth

Utilise la fonction de Paeth avec les pixels en haut, à gauche et en haut à gauche.

Le but est de réduire les valeurs tout en permettant de retrouver les valeurs initiales.

L'encodage avec LZ77 et Huffman (Deflate / Inflate)

PNG utilise un mix de LZ77 et une version modifiée de Huffman pour la compression de données aussi appelé Deflate pour la compression et Inflate pour la décompression.

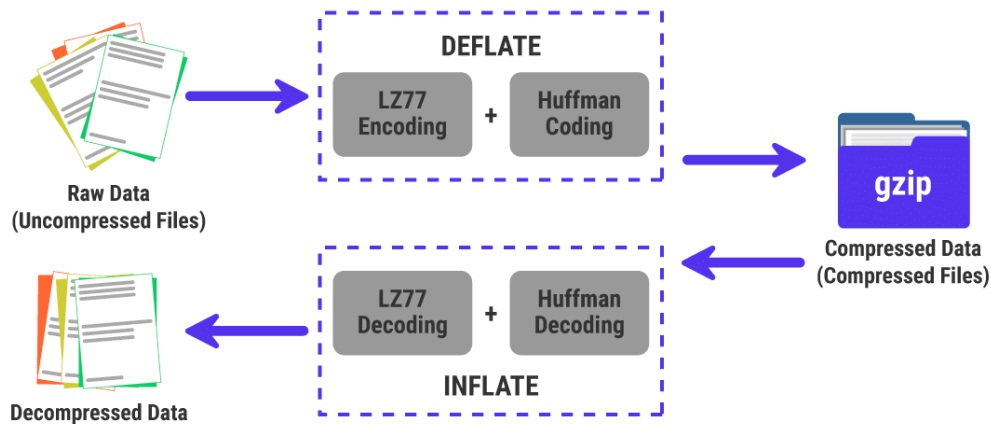


Figure 16 - Cycle de vie Deflate / Inflate

LZ77 est un algorithme qui permet de trouver des séquences égales au sein d'un fichier et lorsqu'une séquence est répétée elle est remplacée par une référence à la séquence initiale.

Cette référence comprend la distance à laquelle la séquence a été rencontrée et sa taille.

LZ77 a deux paramètres, la taille du buffer de recherche et la taille du buffer de "look ahead".

Par exemple :

- buffer de recherche = 15 octets
- buffer de "look ahead" = 4 octets



Figure 17 - Exemple LZ77¹¹

Pour Huffman le principe a été vu plus haut.

La structure d'un fichier PNG

Un fichier PNG est constitué de "chunks" qui ressemblent aux segments JPG dans le principe mais n'ont pas la même structure.

A son minimum un fichier PNG est composé de 4 chunks.

¹¹ <https://www.youtube.com/watch?v=-V48ZygMUtg>

1. La signature PNG, sur 8 octets.
2. Le chunk IHDR qui est l'entête du fichier, sur 25 octets.
3. (Optionnel) Le chunk PLTE qui est ma palette de couleurs du fichier si c'est un PNG sur une profondeur de 8 bits.
4. Le chunk IDAT qui contient les données, d'une longueur égale à la taille des données. Peut exister en plusieurs exemplaires au sein d'un même fichier.
5. Le chunk IEND qui représente la fin de fichier.

D'autres chunks optionnels peuvent aussi exister :

- tIME, l'horodatage de la dernière modification du fichier.
- iTXt, information textuelle internationale.
- tEXt, information textuelle non compressée.
- zTXt, information textuelle compressée.

Voir https://fr.wikipedia.org/wiki/Portable_Network_Graphics pour les autres chunks optionnels.

Si les chunks essentiels sont dans l'ordre précisé ceux-ci peuvent être présents dans n'importe quel ordre.

Chaque chunk suit le format suivant :

1. LENGHT, sur 4 octets.
2. TYPE, sur 4 octets.
3. DATAS, de taille variable
4. CRC, sur 4 octets.

A noter que CRC est généré un algorithme particulier donné sur le lien Wikipédia précédant.

Le chunk d'en-tête est intéressant aussi à préciser car assez spécifique.

Dans l'ordre on trouve au sein de chunks :

1. (DATA) LENGHT, sur 4 octets.
2. TYPE, sur 4 octets, ici 49 48 44 52 en hexadécimal, correspondant donc à IHDR.
3. WIDTH, sur 4 octets.
4. HEIGHT, sur 4 octets.
5. COLOR DEPTH, sur 1 octet.
6. COLOR TYPE, sur 1 octet.
7. COMPRESSION METHOD, sur 1 octet.
8. FILTER METHOD, sur 1 octet.
9. INTERLACING, sur 1 octet.
10. CRC, sur 4 octets.

Hormis les chunks officiels, tout le reste est considéré comme de la DATA.

3.4. L'état de l'art de la stéganographie

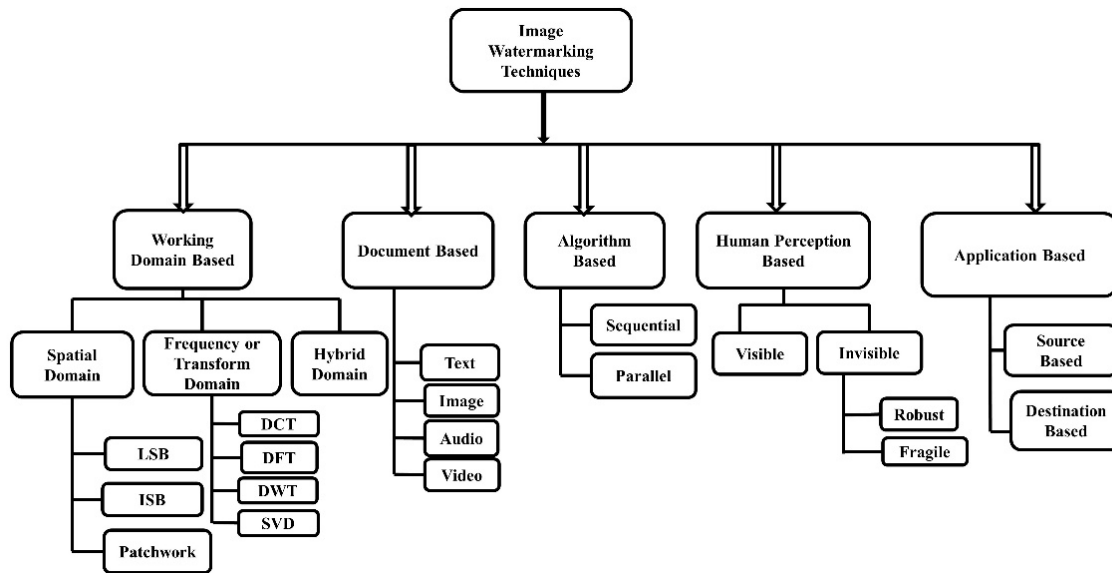


Figure 18 – Vue d'ensemble des différentes techniques de stéganographie

3.4.1. Les domaines d'analyses d'une image

Le **domaine spatial** fait référence à la représentation de l'image telle qu'elle est directement perçue par l'œil humain. Dans ce domaine, l'image est composée de pixels qui sont organisés en une grille régulière. Chaque pixel contient une valeur qui représente l'intensité lumineuse ou la couleur à un emplacement spécifique de l'image. Les opérations effectuées dans le domaine spatial sont appliquées directement sur les pixels de l'image, ce qui permet de manipuler les détails locaux, les contours et les textures de l'image.

Le **domaine fréquentiel**, quant à lui, fait référence à la transformation de l'image dans le domaine des fréquences. Cette transformation permet de représenter l'image en termes de composantes de fréquences et d'amplitudes associées. L'une des transformations fréquentielles couramment utilisées est la transformée de Fourier. Dans le domaine fréquentiel, les caractéristiques globales de l'image, telles que les **variations de luminosité** et **de couleur**, les **motifs récurrents** et les **structures de haute fréquence**, peuvent être analysées et modifiées de manière efficace.

La transformation d'une image du domaine spatial au domaine fréquentiel et vice versa est possible à l'aide de techniques telles que la transformée de Fourier discrète (DFT) ou la transformée en cosinus discret (DCT). Cette transformation permet d'explorer et de manipuler l'image selon des propriétés spécifiques dans le domaine fréquentiel, ce qui peut être avantageux dans certains domaines d'application, tels que la compression d'image ou le traitement du signal.

3.4.2. Domaine spatial¹²

3.4.2.1. Bit de poids faible ou LSB (Least Significant Bit)

La stéganographie LSB (Least Significant Bit) est l'une des techniques les plus simples et couramment utilisées pour dissimuler des informations dans des images. Elle repose sur l'idée d'exploiter les bits de poids faible des pixels de l'image pour encoder les données secrètes.

Dans le contexte de la stéganographie LSB, les images sont généralement converties en niveaux de gris (grayscale) pour simplifier le processus de dissimulation. Cela signifie que chaque pixel est représenté par une valeur de luminosité unique, généralement sur 8 bits (valeurs allant de 0 à 255). Chaque bit des 8 bits représente un niveau de luminosité spécifique, du bit le plus significatif (MSB) au bit le moins significatif (LSB).

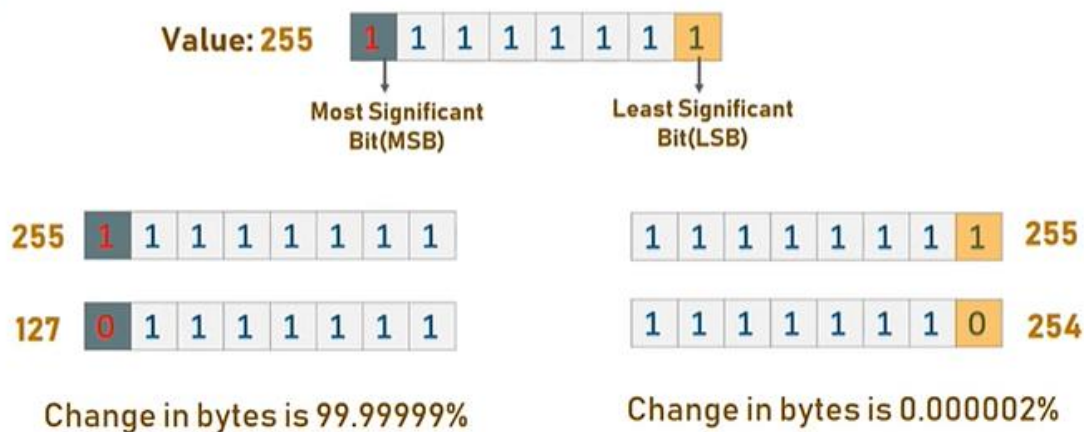


Figure 19 - Représentation MSB et LSB

Pour cacher des données à l'aide de la technique LSB, les bits de poids faible des pixels de l'image sont modifiés de manière à représenter les données secrètes. Par exemple, si vous avez un bit d'information à cacher, il peut être inséré dans le LSB de chaque pixel de l'image. De cette façon, les modifications apportées aux bits de poids faible sont généralement imperceptibles à l'œil humain.

Remarque : il est possible d'utiliser LSB sur une image en couleur. Comme vu précédemment, chaque pixel en couleur est composé de trois canaux de couleur : rouge (R), vert (G) et bleu (B), représentés par des valeurs de 0 à 255. Lors de l'application du LSB sur une image en couleur, les bits de chaque canal de couleur peuvent être utilisés pour cacher des informations. Par exemple, en remplaçant les bits de poids faible (LSB) du canal rouge par les bits du message secret, on peut dissimuler des données sans affecter considérablement l'apparence visuelle de l'image. Cependant, il est important de prendre en compte l'interaction entre les canaux de couleur pour éviter des altérations visibles.

¹² [Hiding data in images using DCT steganography techniques with compression algorithms](#)

Cas d'exemple d'implémentation naïve :

L'insertion séquentielle du message “**lsb**” dans une image couleur de résolution 4x4 pixels

Conversion du message en bit :

En utilisant la table ASCII, nous pouvons convertir le message secret en valeurs décimales, puis en binaire :

l	s	b
01101100	01110011	01100010

Maintenant, nous itérons sur les valeurs de pixel une par une, après les avoir converties en binaire, nous remplaçons chaque bit le moins significatif par les bits du message séquentiellement (par exemple 0100000**0**, nous remplaçons le dernier bit, le bit de droite (**0**) par le deuxième bit de données (**1**) et ainsi de suite). Cela ne modifiera les valeurs de pixel que de +1 ou -1, ce qui n'est pas du tout perceptible.

Encodage :

Utilisateur définit une clé stéganographique, par exemple “MIAGE”

Utilisation d'un algorithme permettant d'obtenir une valeur en fonction du code secret, qui va déterminer l'emplacement de départ du bit dans l'image à partir du quel le message va être inséré (par exemple 108)

Dans notre cas, avec une résolution de 4x4 pixels, nous avons une séquence binaire de $4 \times 4 \times 24 = 384$ bits, donc nous insérerons à partir du 108e bit par rapport à cette séquence.

Insertion des bits du message secret, dans les bits de poids faible des pixels à partir du point de départ.

Insertion d'un marqueur de fin de séquence par exemple une séquence “11111111”.

Décodage :

Le récepteur connaît la clé stéganographique.

La clé est utilisée pour déterminer le point de départ dans l'image où les bits du message secret ont été insérés

Extraction des bits de poids faibles pour reconstituer le message.

Avantages :

- Méthode rapide et facile à mettre en œuvre.
- Très faible altération de l'image d'entrée et difficiles à détecter visuellement.

Inconvénients :

- La résolution de l'image doit être adaptée par rapport à la taille de la donnée à cacher.
- Vulnérable face aux attaques de détection. Les modifications apportées aux bits de poids faible entraînent une répartition statistique différente de celle des images non modifiées. Ainsi, des techniques d'analyse statistique peuvent être utilisées pour détecter la présence de données cachées.

- Capacité de dissimulation limitée. Puisque seuls les bits de poids faible sont modifiés, seule une petite quantité de données peut être dissimulée dans une image donnée sans affecter considérablement la qualité visuelle.
- Faible face à la compression, car les modifications apportées au LSB peuvent être considérées comme du bruit ou de l'irrégularité lors du processus de compression, et peuvent donc être supprimées.
- Le bruit, le filtrage, l'écrêtage, les transformations colorimétriques et le rééchantillonnage sont les points faibles de l'approche LSB.

3.4.2.2. ISB (Intermediate Significant Bit)

La technique ISB, également connue sous le nom de technique de modification des bits intermédiaires, est une approche de stéganographie qui se situe entre la technique LSB et la technique MSB (Most Significant Bit).

Dans la technique ISB, au lieu de modifier directement les bits de poids faible ou de poids fort, on modifie les bits de poids intermédiaire de chaque pixel de l'image. Les bits intermédiaires sont ceux qui ont une position entre les bits de poids fort et les bits de poids faible.

L'idée derrière la technique ISB est d'exploiter le fait que les bits intermédiaires ont une influence moindre sur la perception visuelle de l'image par rapport aux bits de poids fort et de poids faible. En modifiant ces bits intermédiaires, on peut dissimuler des informations sans perturber considérablement l'apparence globale de l'image.

L'avantage de la technique ISB par rapport à la technique LSB est qu'elle offre une meilleure robustesse face aux attaques statistiques. En effet, en modifiant les bits intermédiaires, il est plus difficile pour un attaquant d'analyser la distribution statistique des bits modifiés pour détecter la présence de données cachées.

Cependant, il est important de noter que la technique ISB peut être plus perceptible visuellement que la technique LSB, car les bits intermédiaires peuvent contribuer à des dégradations visuelles plus notables. Par conséquent, il est nécessaire de trouver un équilibre entre la capacité de dissimulation des données et la préservation de la qualité visuelle de l'image lors de l'utilisation de la technique ISB.

3.4.3. Domaine de la transformation

L'intégration dans le domaine de la transformation est une technique d'intégration de données qui utilise des algorithmes spécifiques. Elle est plus puissante que l'intégration dans le domaine spatial. Les systèmes stéganographiques les plus performants fonctionnent généralement dans le domaine de la transformation.

Les techniques du domaine de la transformation ont un avantage par rapport aux techniques LSB car elles cachent les informations dans des zones de l'image qui ne sont pas visibles à l'œil nu. Elles sont également moins susceptibles d'être affectées par la **compression**, le **recadrage** et le **traitement de l'image**. Certaines techniques du domaine de la transformation peuvent être utilisées indépendamment du format de l'image et peuvent même surpasser les conversions de format avec ou sans perte.

Le format de fichier JPEG est le format d'image le plus couramment utilisé sur Internet en raison de sa petite taille pour les images compressées avec ce format.

3.4.3.1. Transformation en cosinus discrète (TCD)

Stéganographie par substitution de bits dans les coefficients de fréquence (LSB) : Cette technique exploite la structure interne du format JPEG qui utilise la transformation de cosinus discrète (DCT) pour la compression des images. L'idée consiste à substituer les bits de poids faible (Least Significant Bits, LSB) des coefficients de fréquence DCT par les bits des données secrètes. Les coefficients de fréquence DCT de l'image sont divisés en blocs de 8x8 pixels, et les valeurs des coefficients sont modifiées de manière subtile pour représenter les données secrètes. Étant donné que les coefficients de fréquence DCT sont plus sensibles aux modifications des bits de poids faible, les modifications apportées ont généralement un impact minimal sur la qualité visuelle de l'image.

3.4.3.2. SVD & IWD

SVD aka Singular Value Decomposition¹³, en Français algèbre linéaire de décomposition, est un outil de factorisation des matrices rectangulaires réelles ou complexes, souvent utilisé dans le traitement d'ondes il peut être appliqué à la stéganographie.

Le IWD aka Integer Wavelet Transform¹⁴ est une représentation d'une fonction d'entier au carré à partir d'une série orthonormée générée par un Wavelet¹⁵, un Wavelet étant une oscillation d'une donnée.

Via le PSNR¹⁶, Peak Signal to Noise Ratio et les 2 méthodes précédentes il est possible de cacher des informations au sein des images. Cette technique permet d'offrir une certaine résistance à la compression mais pas autant qu'il faudrait pour une compression de type JPEG.

¹³ https://en.wikipedia.org/wiki/Singular_value_decomposition

¹⁴ https://en.wikipedia.org/wiki/Wavelet_transform

¹⁵ <https://en.wikipedia.org/wiki/Wavelet>

¹⁶ https://fr.wikipedia.org/wiki/Peak_Signal_to_Noise_Ratio

3.5. Evaluation des différentes techniques

LSB	
Robustesse	Les données cachées dans les bits de poids faible sont sensibles aux perturbations et aux modifications. Des altérations mineures de l'image, telles que le redimensionnement, la compression ou le recadrage, peuvent entraîner la perte ou la corruption des données cachées.
Falsification	Les données cachées utilisant LSB peuvent être facilement altérées ou supprimées par des attaquants malveillants. Si un attaquant identifie la méthode utilisée et connaît l'emplacement des bits de poids faible, il peut falsifier les données cachées ou les éliminer complètement.
Attaques géométriques	Les transformations géométriques appliquées à l'image, telles que la rotation, la translation ou la transformation affine, peuvent modifier la position des bits de poids faible, entraînant ainsi la perte des données cachées ou leur détection plus facile.
Filtrage	Certains algorithmes de filtrage, utilisés pour améliorer la qualité de l'image ou réduire le bruit, peuvent supprimer les informations cachées dans les bits de poids faible. Les filtres linéaires et non linéaires peuvent altérer les valeurs des pixels et éliminer les données cachées.
Compression	Lorsque l'image subit une compression avec des techniques de compression avec pertes, les bits de poids faible peuvent être modifiés ou supprimés, entraînant une perte partielle ou totale des données cachées. Les algorithmes de compression tels que JPEG sont connus pour leur impact négatif sur la stéganographie LSB.

3.6. Choix de l'implémentation finale

Pour l'implémentation finale j'ai choisi une approche basé TCD en modifiant les derniers bits de pixels, ce qui permet de limiter les dégâts sur la qualité de l'image avec un pattern aléatoire afin de rendre le message difficile à lire pour ceux qui ne l'ont pas produit.

Cette solution ne s'applique qu'aux fichiers PNG car les fichiers JPEG peuvent potentiellement remplacer tous les pixels d'une image ce qui rend mon approche impossible à appliquer.

Afin d'implanter cela je vais server side, donc ici via Python Flask, implémenter un algorithme qui prend en paramètre une image et la y implémenter un message.

Du côté du frontend je vais utiliser React dû à sa rapidité, flexibilité et afin de personnellement m'améliorer à ce Framework avec Material UI pour faciliter la création de l'UI.

Material UI fourni des Templates prêtes à l'emploi ce qui réduit le temps de développement à l'UI.

Un serveur PostgreSQL sera mis en place pour sauvegarder les données utilisateur, et Amazon AWS sera utilisé pour stocker des images via Buketeer

4. Réalisation

4.1. Technologies utilisées

4.1.1. JavaScript¹⁷

Javascript est un langage de scripting côté client qui tourne sur les navigateurs. Il permet de coder les interactions au sein d'un site web ou même avec Node.js¹⁸ un serveur à part entière.

Dans le cadre de ce projet Javascript ne sera utilisé que pour du frontend c'est-à-dire côté client.

4.1.2. Python¹⁹

Python lui aussi est un langage qui beaucoup d'usage du frontend, côté client au backend au sein d'un serveur. Sa simplicité permet de manipuler facilement des images ce que dans le cadre de ce projet d'avère très utile.

4.1.3. React²⁰

React est un Framework Javascript qui permet de coder des sites qui tourneront côté client. Son avantage est sa rapidité et sa souplesse par à rapport à ses concurrents.

React permet aussi de faire du cross-platform avec React Native ce qui pourrait d'avérer utile pour des objectifs ultérieurs à ce projet.

React permet de coder en Javascript ou TypeScript, dans le cadre du projet seulement du Javascript sera utilisé en raison des temps de compilation TypeScript ainsi que la simplicité syntaxique de Javascript comparé à TypeScript. Si jamais le projet se devait être cross-platform alors une réécriture en TypeScript serait nécessaire. React fourni dès la création d'un projet une structure qui supporte les tests ce qui simplifie le processus de testing de l'application.

¹⁷ <https://developer.mozilla.org/fr/docs/Learn/JavaScript>

¹⁸ <https://nodejs.org/fr>

¹⁹ <https://www.python.org/>

²⁰ <https://react.dev/>

4.1.4. Flask²¹

Flask est un Framework Python qui donne une syntaxe simple dans le but de créer une API ou même un site web entier.

Flask donne accès à des outils de test simplement également.

Dans le cadre du projet seule son utilité API sera utilisée.

4.1.5. PostgreSQL²²

PostgreSQL est une base de données SQL open source qui fournit tous les avantages SQL, c'est-à-dire les principes ACID :

- Atomicité
- Consistance
- Isolation
- Durabilité

Ces principes permettent de garantir que les données ne sont pas corrompues lors des transactions et sont correctes tout le temps. Cela est important pour ce projet en raison du stockage associés aux identifiants présents sur les images signées.

De plus le service de hosting choisi, Heroku²³ propose une intégration PostgreSQL.

4.1.6. Bucketeer²⁴

Bucketeer permet de faire le lien avec Amazon AWS S3 afin de stocker des images et est aussi intégré à Heroku.

4.2. Logiciels utilisés

4.2.1. Visual Studio Code

VS Code est un des meilleurs éditeurs de texte quand il s'agit de travailler avec React grâce aux extensions disponibles ce qui en fait un choix intéressant pour ce projet.

Il permet aussi de travailler avec Python ce qui permet de n'utiliser qu'un éditeur de texte pour tout le projet.

4.2.2. PyCharm

PyCharm est un IDE spécialisé pour Python ce qui peut s'avérer utile dans des cas complexes du projet.

²¹ <https://flask.palletsprojects.com/en/2.3.x/>

²² <https://www.postgresql.org/>

²³ <https://www.heroku.com/what>

²⁴ <https://devcenter.heroku.com/articles/bucketeer>

4.2.3. TablePlus²⁵

TablePlus est un logiciel qui permet de visualiser et insérer des données dans une longue liste de base de données disponibles, PostgreSQL inclus.

4.2.4. Postman²⁶

Postman permet de tester des requêtes facilement sans frontend prêt, ce qui permet de tester facilement les requêtes vers l'API Flask.

4.3. Outils de gestion de projet

4.3.1. Git²⁷

Git est un outil de versioning open source qui permet donc de créer plusieurs versions d'un même projet dans le temps. Cela permet de toujours pouvoir revenir à un point où le projet est stable en cas de problème.

Git offre aussi d'autres fonctionnalités comme les branches qui permettent par exemple d'avoir plusieurs branches de développement tel qu'une branche de développement active est une version stable figée dans le temps pour déployer une application.

4.3.2. GitHub²⁸

GitHub est un service qui permet d'héberger des git sur un serveur distant en plus d'autres fonctionnalités tel que l'affichage de statistiques intéressantes.

4.3.3. Trello

Trello est un service qui fournit des Kanban et permet de les partager avec son équipe.

En début de projet lorsque qu'on était deux sur le projet il avait été prévu d'utiliser Trello pour se partager les tâches mais lorsque j'ai fini seul j'ai décidé de ne pas utiliser Trello car j'évalue que Trello n'avait aucun intérêt sans partage de tâches.

²⁵ <https://tableplus.com/>

²⁶ <https://www.postman.com/>

²⁷ <https://git-scm.com/>

²⁸ <https://github.com/>

4.4. Architecture

Ci-dessus ont peut voir l'architecture de l'application en entier.

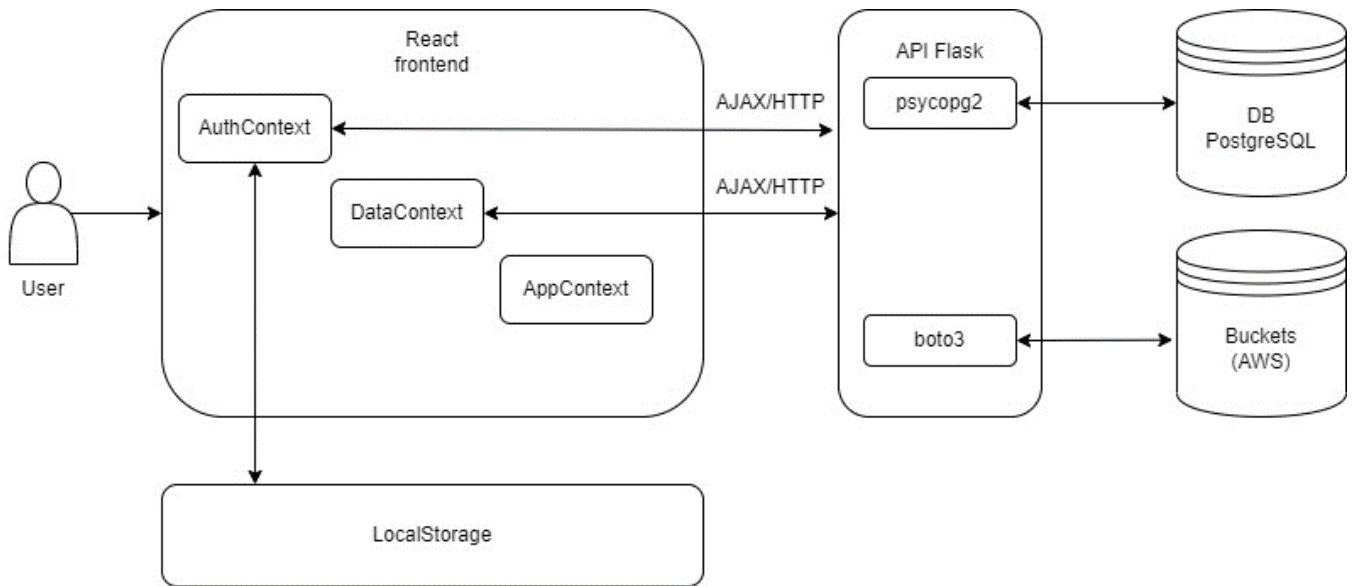


Figure 20 – Architecture de l'application

Description du schéma :

Le client frontend en React est l'interface présenté à l'utilisateur.

Celui-ci communique avec le backend via des Contextes propres a React qui seront décrits plus tard.

Le contexte AuthContext utilise le LocalStorage afin de s'assurer une permanence des données en cas de refresh de la page en raison du fait que toutes les données de React sont provisoires.

Les échanges frontend/backend se font en AJAX côté frontend et via des requêtes HTTP côté backend.

Le backend est une API REST qui fonctionnent donc avec des endpoints.

Un endpoint se présente comme la figure suivante.

```
@app.route('/api/0.1/user/<mail>')
def getUserInfo(mail):
    mail = escape(mail)
    cursor = co.cursor()
    cursor.execute("SELECT id, username, mail FROM users WHERE mail = '{mail}'".format(**locals()))
    data = cursor.fetchone()
    print(data)
    userdata = {'id': data[0], 'username': data[1], 'mail': data[2]}
    return userdata
```

Figure 21 – Exemple de endpoint Flask (Python)

L'API est connectée à une base de données PostgreSQL et des Buckets²⁹ sur AWS.

La base de données sert à stocker les informations utilisateurs principalement et AWS à stocker des images dans des « Buckets ». Chaque Bucket permet de « ranger » les images et dans le cadre de ce projet a les associer à un ID utilisateur.

Ces connexions à ces 2 services se font via les connecteurs psycopg2³⁰ pour la DB et boto3³¹ pour AWS.

²⁹ <https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingBucket.html>

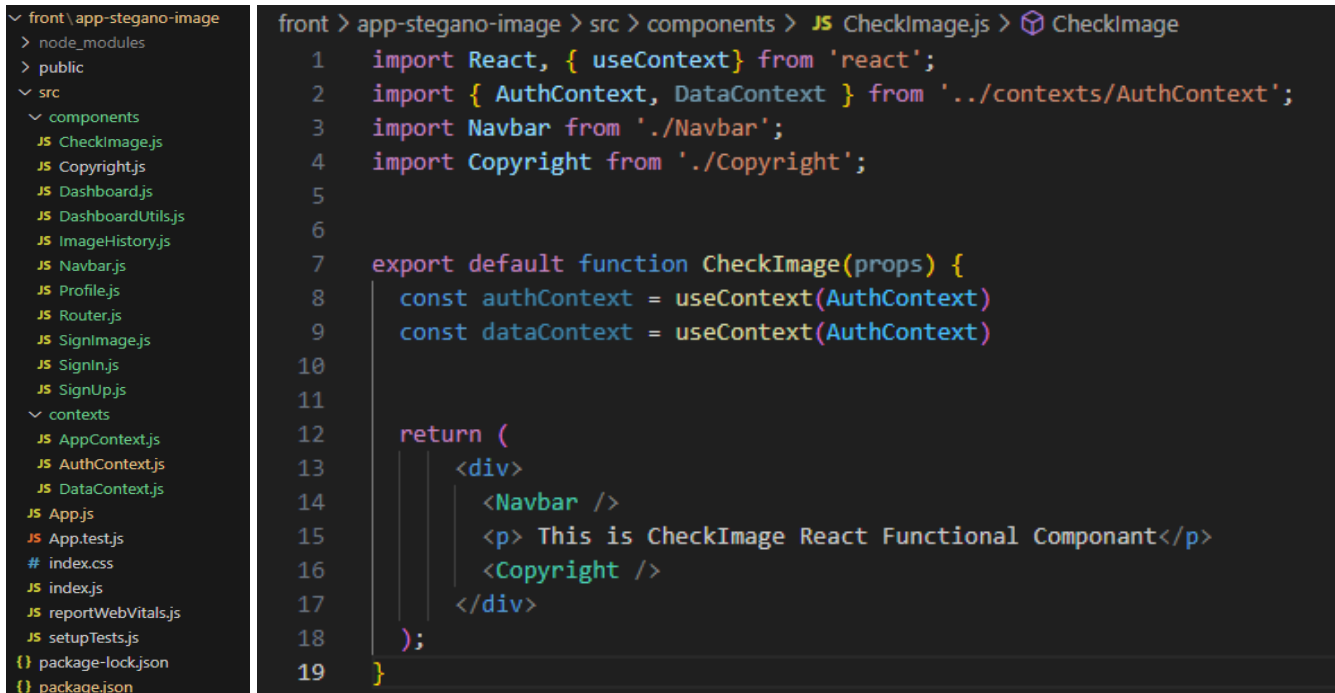
³⁰ <https://pypi.org/project/psycopg2/>

³¹ <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

4.5. Le client frontend

Le client frontend est un projet React.

React est assez flexible et peut-être utilisé de différentes manières.



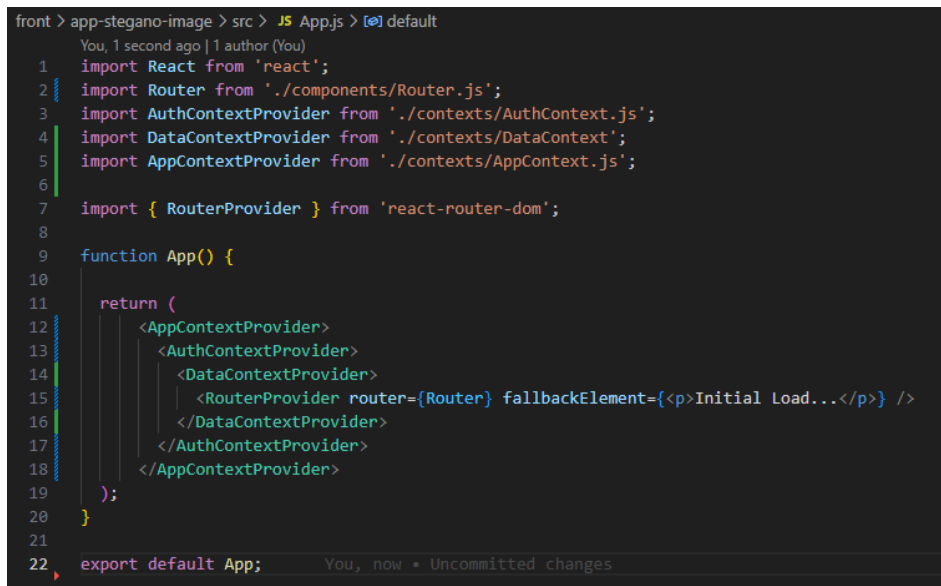
The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the project structure: front > app-stegano-image > src > components. The code editor shows the content of CheckImage.js, which is a simple functional component using React and Context API.

```
front > app-stegano-image > src > components > JS CheckImage.js > CheckImage

1  import React, { useContext } from 'react';
2  import { AuthContext, DataContext } from '../contexts/AuthContext';
3  import Navbar from './Navbar';
4  import Copyright from './Copyright';
5
6
7  export default function CheckImage(props) {
8    const authContext = useContext(AuthContext)
9    const dataContext = useContext(AuthContext)
10
11
12    return (
13      <div>
14        <Navbar />
15        <p> This is CheckImage React Functional Component</p>
16        <Copyright />
17      </div>
18    );
19  }
```

Figure 22 – Structure du projet React et exemple d'un Component simple (non fini)

Comme on le voit ici React est composé d'un ensemble de Component dont le root est App car le premier chargé, donc App est le père de tous les autres Components. Ces Components peuvent être des classes ou des fonctions, cela dit l'approche fonctionnelle est encouragé aujourd'hui.



The screenshot shows a code editor with the content of App.js. The component is a functional component that uses React, Router, and Context API to provide context to other components.

```
front > app-stegano-image > src > JS App.js > default

You, 1 second ago | 1 author (You)
1  import React from 'react';
2  import Router from './components/Router.js';
3  import AuthContextProvider from '../contexts/AuthContext.js';
4  import DataContextProvider from '../contexts/DataContext.js';
5  import AppContextProvider from '../contexts/AppContext.js';
6
7  import { RouterProvider } from 'react-router-dom';
8
9  function App() {
10
11    return (
12      <AppContextProvider>
13        <AuthContextProvider>
14          <DataContextProvider>
15            <RouterProvider router={Router} fallbackElement={<p>Initial Load...</p>} />
16          </DataContextProvider>
17        </AuthContextProvider>
18      </AppContextProvider>
19    );
20  }
21
22  export default App; You, now • Uncommitted changes
```

Figure 23 – Le Component App

Typiquement dans un projet simple React App ne contiendrait que les routes, mais j'ai choisi une approche avec des Contexts³².

Cette approche permet d'avoir à disposition des variable et fonctions

Une Context est un Component React qui ne sert pas à afficher une page mais à contenir des données qui seront accessibles et modifiables par les fils de ce Context en les englobant dans un ContextProvider fourni par le contexte en question. Ces ContextProviders sont visibles sur la figure 23.

Sur la figure ci-dessous on peut voir le Contexte AppContext choisi pour sa simplicité.

```
front > app-stegano-image > src > contexts > JS AppContext.js > AppContextProvider
1  import React, { Component, createContext } from "react";
2
3  export const AppContext = createContext()
4
5  export default class AppContextProvider extends Component {
6      constructor(props) {
7          super(props)
8          this.state = {
9              currentPage: "SignIn"
10         }
11     }
12
13     render() {
14         return (
15             <AppContext.Provider value={{ ...this.state, }}>
16                 {this.props.children}
17             </AppContext.Provider>
18         )
19     }
20 }
```

Figure 24 – Le Context AppContext

On peut voir que les Contexts fonctionnent via le mécanisme des props, aussi présent sur Angular.

Dans la fonction « render() » il est retourné le Context avec comme props « this.state » qui est défini dans le constructeur et contiendra toutes les propriétés, fonctions et méthodes que les fils doivent accéder.

Les fils sont contenus dans « this.props.children ».

Alors on pourrait se poser des questions, pourquoi choisir d'utiliser un mécanisme plus complexe à mettre en place que des props simples ? La réponse est plusieurs avantages :

- Une encapsulation contrôlée.
- Des props qui ne sont pas en mode read-only dans les fils.
- La possibilité de partager les méthodes souvent utilisées comme celles liés à l'authentification sur tous les fils.
- Une meilleure façon de structurer le code, je n'ai qu'à modifier une méthode dans AuthContext plutôt que partout où elle serait présente dans la code base.

³² <https://react.dev/reference/react/createContext>

Maintenant que les sous-parties sont présentées, ci-dessous la structure générale du projet React est visible.

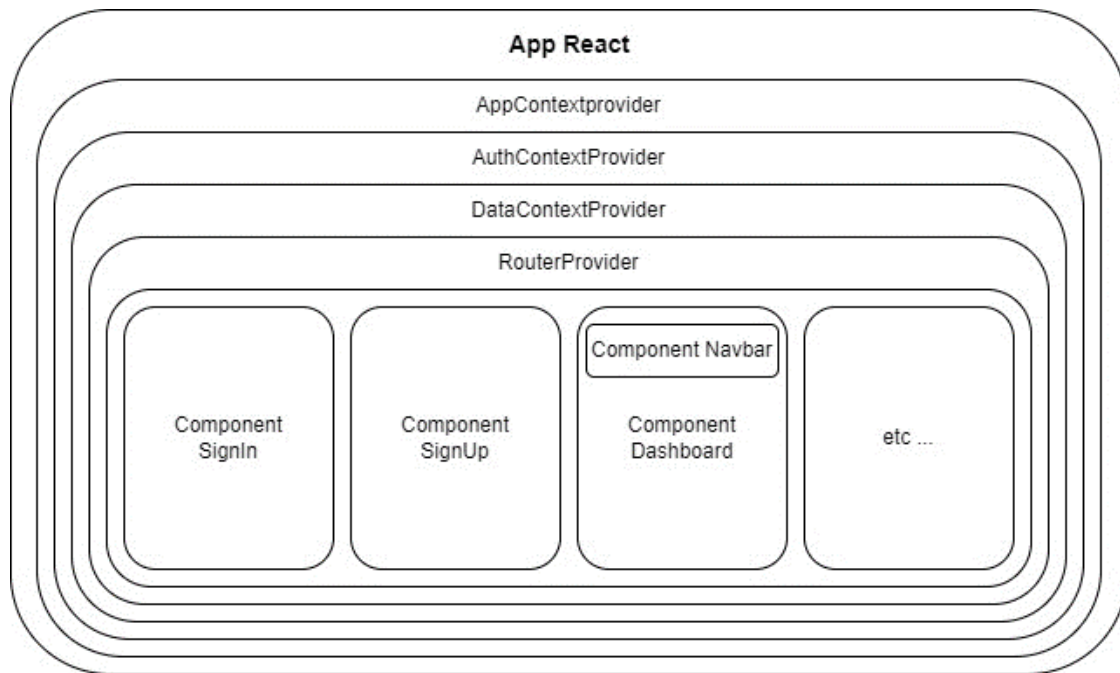


Figure 25 – L'architecture du projet React

Certaines parties restent cependant à définir ainsi que certains problèmes non résolus.

Le premier est le **RouterProvider**. Il s'agit d'une des façons que React fournit pour créer un Router.

Son avantage est la raison pour laquelle j'ai choisi ce Router au lieu d'un autre plus simple est la possibilité via une fonction de faire des tests avant d'afficher ce que la Route devrait retourner par défaut.

Dans la figure 23, ligne 15, on peut voir « router = {Router} ».

Cela indique que le RouterProvider doit utiliser l'object Router comme base le Router de l'application.

```
const router = createBrowserRouter([
  {
    id: "root",
    path: "/",
    loader() {
      return 'loader';
    },
    Component: Layout,
    children: [
      {
        index: true,
        element: <SignIn />,
      },
      {
        path: "signin",
        // action: privateAction,
        loader: inviteLoader,
        element: <SignIn />,
      },
      {
        path: "signup",
        // action: publicAction,
        loader: inviteLoader,
        element: <SignUp />,
      },
      {
        path: "dashboard",
        // action: publicAction,
        loader: userLoader,
        element: <Dashboard />,
      },
    ],
  },
]);
```

Figure 25 – L'object Router

Ci-dessus je défini un Router en Javascript au lieu de le faire en JSX. On voit que pour chaque route je peux définir un loader. Un loader est une fonction qui s'exécute avant l'affichage du Component associé à la route. L'idée est que de cette façon je peux limiter les accès à certaines routes selon ce que dit AuthContext.

Finalement on peut voir un problème sur la figure 25, pourquoi le component SignIn a accès aux données de DataContext qui d'adresse au transfert de fichiers. La raison est que c'est Router qui donne accès aux composants dans cette application et je n'ai pas trouvé de solution pour limiter l'accès aux différents Context dans les délais impartis. Une approche serait peut-être différents Routers.

J'espérais pouvoir limiter les accès à certaines routes à partir de ce composant mais je n'y suis pas parvenu encore.

4.6. L'API backend

Au niveau du backend le projet se base donc sur une API Python via le Framework Flask.

Flask permet de mettre en place des endpoints facilement ce qui est parfait pour une API.

En termes de structure Flask est libre, il est possible de travailler avec un seul fichier mais pas en termes de maintenance de code et de test.

J'ai donc adopté la structure à partir d'une vidéo tutorial sur YouTube³³.

La vidéo fourni tous les éléments importants y compris pour pouvoir tester l'application.

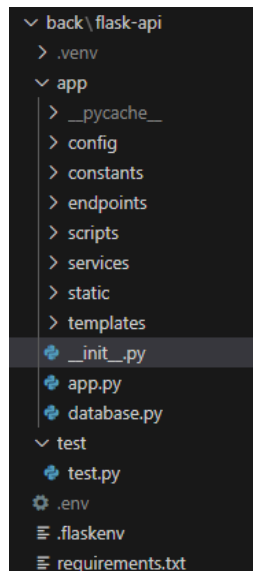


Figure 26 – La structure du code Flask

L'application se lance via `__init__.py`, ce fichier configure l'object app qui est l'application.



Ce fichier indique aussi un module Python donc on peut considérer toute l'application comme un module Python.

Les autres éléments importants sont :

- config : Ou les fichiers de configurations seront présents.
- endpoints : Ou seront les fichier contenant tous les endpoints, permet un code plus lisible.
- scripts : Ou seront les scripts notamment l'implémentation de la stéganographie.
- database.py : Configure la connection à la base PostgreSQL via flask_sqlalchemy.
- test : Ou se trouveront les fichiers de test.
- .flaskenv : fichier contient les variables d'environnement envoyé aussi à Github, informations non confidentielles.
- .env : fichier contenant les variables d'environnement confidentielles .

³³ <https://www.youtube.com/watch?v=WFzRy8KVcrM>

```

back > flask-api > app >  __init__.py > ...
    You, 22 hours ago | 1 author (You)
1  from flask import Flask
2  from flask_cors import CORS
3  import os
4
5  from app.endpoints.auth import auth
6  from app.endpoints.files import files
7  from app.endpoints.user import user
8
9  from app.scripts.lsb import lsb
10  You, 22 hours ago * 10k refactorings later + start of image changing...
11 from app.database import db
12
13 # Done with https://www.youtube.com/watch?v=WFzRy8KVcrM
14
15 def create_app(test_config = None):
16
17     app = Flask(__name__, instance_relative_config = True)
18     cors = CORS(app)
19     #app.config['CORS_HEADERS'] = 'Content-Type'
20
21     if test_config is None:
22         app.config.from_mapping(
23             SECRET_KEY = os.environ.get("SECRET_KEY"),
24             SQLALCHEMY_DATABASE_URI = os.environ.get("SQLALCHEMY_DATABASE_URI"),
25             SQLALCHEMY_TRACK_MODIFICATIONS = False,
26             CORS_HEADERS = 'Content-Type'
27         )
28
29     else:
30         app.config.from_mapping(test_config)
31
32     db.app = app
33     db.init_app(app)
34
35     app.register_blueprint(lsb)
36
37     app.register_blueprint(auth)
38     app.register_blueprint(files)
39     app.register_blueprint(user)
40
41     return app

```

Figure 27 – code de __init__.py

Ce fichier contient la création du fichier app et sa configuration.

La configuration passe par des variables d'environnement mais devrait à la fin être importé d'un fichier de config du répertoire config.

Ce fichier lie app à d'autres fichiers comme auth, ou lsb, cela permet de séparer le code en plusieurs fichier via les Blueprints Python.

```
auth = Blueprint('auth', __name__, url_prefix= '/api/0.1/auth')
```

Figure 28 – Déclaration d'un Blueprint Python

CORS permet d'accepter certaines requêtes utilisées du côté frontend.

```

@auth.post('/updateuser')
def updateUser():
    id = request.json['id']
    username = request.json['username']
    email = request.json['email']

    if(" " in username):
        return {'error': 'Username must be alphanumeric and not contain spaces.'}, HTTP_400_BAD_REQUEST

    if(not validators.email(email)):
        return {'error': 'Email is not valid.'}, HTTP_400_BAD_REQUEST

    if(User.query.filter(email == email, id != id).first() is not None):
        return {'error': 'Email is already in use.'}, HTTP_409_CONFLICT

    user = User.query.filter_by(id = id).first()

    if(user is not None):
        user.username = username
        user.email = email
        db.session.commit()
        return {'message': 'User updated with success.'}, HTTP_200_OK
    else:
        return {'error': 'User with id %scould not be found.'%(id)}, HTTP_400_BAD_REQUEST

```

Figure 29 – Un endpoint (/updateuser)

Un endpoint se présente comme ceci.

- « .post » permet de savoir qu'il s'agit d'une requête http POST, pour get « .get » peut être utilisé.
- « requete » représente l'object reçu et une fonction comme « json() » ici est appliqué dessus afin d'avoir des informations accessibles. A noter que tous les échanges se font au format JSON ce qui simplifie les choses.
- « flask_sqlalchemy » permet d'interroger la base de données facilement à partir de modèles vu plus tard. Le modèle est appelé par « User » suivi de « query » pour requête et enfin « filter » pour un select.
- Le « return » est la réponse reçu par le frontend que j'ai standardisé sur le modèle suivant :
 - ❖ {'message' : msg, 'objs_à_envoyer' : objs}, http_200(ish) en cas de réussite.
 - ❖ {'error' : err}, http_400ish en cas d'erreur contrôlé.
 - ❖ Cela permet côté front d'utiliser le « message » ou « error » comme status.

```

if (request.status === 200) {
  this.setState(prevState => ({ ...prevState, email: updateData.email }));
  this.setState(prevState => ({ ...prevState, username: updateData.username }));
  this.updateLocalStorage(this.state.token, this.state.id, updateData.email, updateData.username, this.state.created_at)
  console.log(requestJSON.message)
  return { message: requestJSON.message, confirmation: true, code: request.status }
}
else {
  console.log(requestJSON.error)
  return { message: requestJSON.error, confirmation: false, code: request.status }
}

} catch (err) {
  console.log(err)
  return { error: err.message, confirmation: false, code: "None" }
}
}

```

You, 23 hours ago • 10k refactorings laters + start of image changing..

Figure 30 – Traitement de la réponse d'un requête frontend

Du côté front end je revoie aussi au composent visuels toujours le même Object autant que possible.

{ error/message : err/msg, confirmation: true/false, code: status_de_la_requête http }

Cela permet de généraliser un peu le code et ne pas avoir à faire des aller-retour entre fichiers pour vérifier les types de retour.

```

back > flask-api > app > endpoints > files.py > upload_files
You, 1 second ago | 1 author (You)
1  from flask import Blueprint, request
2
3  from app.scripts.lsb import lsb
4  from app.scripts.aws import uploadToAWS
5
6  from app.constants.httpStatusCodes import *
7
8  files = Blueprint('files', __name__, url_prefix= '/api/0.1/files')
9
10 @files.post('/upload')
11 def upload_files():
12     req = request.json()
13     files = req.files
14     id = req.id
15
16     for filename, file in files:
17         file.save(filename)
18         # print(filename, file)
19         newFile = lsb(file)
20         uploadToAWS(id, file)
21
22     return { 'message': 'File processed', file: newFile }, HTTP_200_OK

```

Figure 31 – Requête /upload

Ce endpoint permet de traiter un, ou des images via la fonction « lsb », de les envoyer à un Bucket AWS S3 via la fonction « uploadToAWS » avec comme nom l'id de m'utilisateur et de renvoyer au frontend les images traitées.

C'est également du code récent et donc pas une version finale.

```

class User(db.Model):
    id = db.Column(db.Integer, primary_key = True)
    username = db.Column(db.String(50), unique = True, nullable = False)
    salt = db.Column(db.String(50), unique = True, nullable = False)
    hashpass = db.Column(db.String(50), unique = True, nullable = False)
    email = db.Column(db.String(50), unique = True)
    created_at = db.Column(db.DateTime, default = datetime.now)
    updated_at = db.Column(db.DateTime, onupdate = datetime.now)
    image_keys = db.relationship('ImageKey', backref = 'user', lazy = True)

    def __repr__(self):
        return '<User %r>' %self.username

```

Figure 32 – Le modèle User flask_sqlalchemy

Les modèles « sqlalchemy » permettent de créer des modèles à l'image des tables et de les utiliser directement comme lien à la base de données.

« __repr__(self) » permet de s'assurer qu'une seule instance existe.

Ce modèle met aussi automatiquement à jour « updated_at » sans à avoir à le faire manuellement ainsi que « created_at » via la méthode « default ».

5. Conclusion

5.1. Tâches effectuées

J'espère pour le 29/08 avoir fait les tâches en vert sur le tableau suivant qui correspond aux exigences initiales.

<ul style="list-style-type: none">• Produire une analyser complète des différentes techniques et implémentations existantes de la stéganographie qui offrent une résistance importante à la compression.• Choisir avec justification une des implémentations analysées auparavant.
Construction de la plateforme de signatures d'images
<ul style="list-style-type: none">• Mettre en place un système d'inscription sécurisé.• Mettre en place un système d'identification sécurisé.• Mettre en place une fonction d'upload d'images qui seront automatiquement signés pour les utilisateurs connectés.• La signature d'images doit être résistante à la compression.• La signature d'images doit altérer le moins possible la qualité de l'image originale.• Mettre en place une page où l'utilisateur peut upload une image et obtenir les informations de l'auteur de l'image via une vérification de la signature.• L'application doit tenir compte des meilleures pratiques en matière de sécurité en utiliser des mécanismes d'authentification robustes.• La confidentialité des données utilisateurs doit être assuré.• Le front est libre au niveau de l'implémentation.• Le backend doit être en Flask, un Framework Python.

Cela correspond au MVP et donc pour cela je suis content.

J'espère pouvoir également déployer le code sur Heroku pour la démonstration le 29/08.

5.2. Perceptives futures

Hormis les autres exigences beaucoup reste à faire et à améliorer.

Voici une liste non exhaustive :

Pour le frontend :

- Refactor le frontend entièrement en React fonctionnel.
- Trouver une solution à l'asynchronisme des `useState()` de React.
- Créer un UI totalement adapté aux mobiles, l'actuel est plutôt responsif mais inadapté.
- Le contrat utilisateur pour les cookies.
- Tester l'application avec des test automatisés que je n'aurais pas temps de faire pour le 29 /08, le frontend et le backend devrait déjà être prêts pour cela.
- Interdire des routes aux utilisateurs non connectés, j'ai eu des soucis au niveau de React pour accomplir ce but et pas le temps de le finaliser.

Pour le backend :

- Limiter l'accès à certains endpoints, le tutorial couvre cela.
- Permettre un accès public a certaines endpoints.
- Améliorer l'algorithme d'insertion de message dans les images pour le moment c'est un peu long pour une version de production.
- Utiliser des couples clés publiques, clés privées pour identifier les images afin d'avoir une meilleure sécurité.

En général :

- Avoir des serveur API, PostgreSQL et Bucketeer redondants afin d'assurer une disponibilité permanente et faire des changements dans le code pour rendre cela possible.

En termes de fonctionnalités futures :

- Pouvoir se connecter via Google, Apple etc.
- Fournir un component pour afficher ses images sur d'autres sites.
- Des liens publics aux images afin de les partager facilement.
- Un board public pour tous les utilisateurs de l'app pour les images souhaitées, par default les images seraient privées :
 - ❖ Dans ce board les utilisateurs pourraient commenter les images, les ajouter en favori.

L'application tel qu'elle est maintenant est juste un MVP donc elle doit forcément encore être amélioré via les fonctionnalités des listes précédentes ou autres.

5.3. Difficultés rencontrées

Le projet c'est avéré être difficile en termes technique et en termes de gestion de temps. Cela se voit par la date d'envoi de ce rapport.

J'ai sous-estimé le temps de recherche en premier lieu qui s'est avéré difficile en raison des sujets recherchés et abordés.

En termes de développement je n'ai jamais fait de projets sérieux en React auparavant.

Par conséquent certaines mécaniques comme le life-cycle de React m'échappaient et m'ont mené sur des fausses routes qui m'ont fait perdre beaucoup de temps en raison de refactorings successifs qui ne fonctionnaient pas comme prévu.

Même maintenant je ne sais pas comment m'assurer que « `useState()` » ait attribué bonnes valeurs aux variables du state au bon moment ce qui mène à l'exécution de méthodes qui n'ont pas encore les informations actualisés.

En ce qui concerne le backend si l'on use Flask de façon basique c'est assez simple à prendre en main et à utiliser.

Mais une utilisation plus avancée requiert une maîtrise de Python que je n'avais pas non plus ce qui m'a aussi fait perdre du temps, heureusement le tutorial mentionné précédemment m'a aidé à créer un projet avec une structure correcte et à travers le tutoriel j'ai découvert de nombreuses mécaniques de Python.

5.4. Bilan personnel

Personnellement j'ai appris beaucoup de choses que ce soit au niveau de la recherche sur la stéganographie que des technologies utilisées. Utiliser React comme frontend avec des lacunes était certainement un défi mais malgré le temps perdu je pense que les notions apprises seront forcément utiles dans le futur.

Même si j'ai mal géré le projet au niveau du temps je suis content d'avoir atteint un MVP, et je suis content d'avoir produit un projet full stack.

Je pense continuer le projet suite au 29/08 afin de finir au moins certaines fonctionnalités que je n'ai pas eu le temps de mettre en place et utiliser toute la structure de la partie backend Python dont je n'ai pas eu l'occasion de profiter encore.

Pour conclure je suis content du résultat malgré la mauvaise gestion du temps.