

PROJET TUTORÉ

MASTER 1 INFORMATIQUE
ANNÉE 2024-2025

Outils de génération automatique de patron de broderie au point de croix.

MARRAS Tristan, BAUD Joris, BOUPACHA Sid Ali, TYLL Jürgens, RABHI Massissilia, BOUDJELI Ines, MORTIER Marie, RIANE Lysia, FOUCHARD Johann, STRAINCHAMPS Clothilde, MESLEM Zohra

Encadré par
MIGNIOT Cyrille



Decembre 2024

Table des matières

1 Contexte du projet et objectifs	2
1.1 Présentation du projet	2
1.2 Objectifs globaux	2
2 Organisation de l'équipe	3
2.1 Rôles attribués	3
2.2 Réunions et coordination	3
2.3 Évolution de l'organisation	4
3 Répartition effective du travail	5
3.1 Tâches initialement prévues	5
3.2 Tâches effectivement réalisées	6
3.3 Analyse des écarts	7
4 Fonctionnalités développées	8
4.1 Le développement du site web	8
4.2 Conversion d'image en patron (SCRIPT)	12
4.3 Stockage et gestion des patrons	17
4.4 Tâches réalisées par l'équipe DevOps	19
5 Conclusion et perspectives	27
5.1 Adéquation des résultats obtenus	27
5.2 Améliorations et objectifs pour les semestres suivants	27
5.3 Mot de la fin	27
A Annexes	29

1 Contexte du projet et objectifs

1.1 Présentation du projet

Dans le cadre de la 1ère année de master informatique à l'université de Bourgogne, nous avons été amenés à former un groupe d'étudiants et à choisir un sujet pour l'enseignement de 'projet tutoré'. Le projet vise à développer une application web permettant de créer automatiquement des patrons de broderie au point de croix à partir d'images. Ce processus, inspiré du fonctionnement des pixels dans une image, repose sur la transformation de chaque carré d'une grille en une croix brodée, associée à une couleur de fil spécifique. L'outil devra non seulement générer des patrons adaptés aux besoins des brodeurs, mais aussi proposer des fonctionnalités pratiques telles que la gestion des paramètres de génération, l'optimisation et la clusterisation des couleurs, ainsi que le stockage des patrons produits.

L'objectif est de fournir un service qui simplifie la conception d'un patron de broderie en le rendant accessible à un large public.

1.2 Objectifs globaux

L'application doit répondre à plusieurs objectifs pour répondre au cahier des charges du projet et garantir sa pertinence auprès des utilisateurs :

- **Création automatique de patrons** : transformer une image en un patron exploitable, tout en minimisant les ajustements manuels nécessaires.
- **Adaptabilité des paramètres** : permettre aux utilisateurs de configurer des options, telles que la dimension de la grille ou le nombre de couleurs, afin de personnaliser le résultat en fonction de leurs préférences ou contraintes.
- **Simplification de la broderie** : réduire les changements de fils inutiles grâce à une optimisation des zones de couleurs homogènes, en s'appuyant sur des algorithmes comme k-means.
- **Standardisation des couleurs** : garantir une correspondance entre les couleurs générées et les références du nuancier DMC, facilitant ainsi la sélection des fils nécessaires.
- **Gestion des ressources** : fournir des informations pratiques, telles que la quantité estimée de fil nécessaire pour chaque couleur.
- **Facilité de stockage et de gestion des patrons** : permettre la sauvegarde des patrons générés, afin de les réutiliser ainsi que les partager à la communauté du site.
- **Accessibilité pour tous** : développer une interface simple, intuitive, adaptée aussi bien à des utilisateurs novices que confirmés.

2 Organisation de l'équipe

2.1 Rôles attribués

2.1.1 Responsable de groupe

Joris est chargé de coordonner le groupe, de s'assurer du respect des délais, et de faciliter la communication entre les membres. Il veille à l'avancement global du projet et à la résolution des problèmes éventuels.

2.1.2 Devops

Ce rôle est occupé par Jürgens, avec l'aide de Clothilde. Ils sont responsables de la gestion des outils d'intégration et de déploiement, ainsi que de l'infrastructure nécessaire pour le projet.

2.1.3 Référent données

Lysia gère les aspects liés aux données : leur collecte, leur traitement, et leur gestion. Elle veille à ce que les données utilisées soient correctes et à jour et stocké dans un environnement adapté. Elle est accompagnée de Massissilia et Ines.

2.1.4 Référent développeur

Sid Ali s'assure de la qualité et de l'efficacité du développement et veille la cohérence globale du code. Il est le point de référence pour les questions techniques et de programmation. Il est aidé de Joris pour le développement web et de Tristan, Marie, Johann et Zohra pour la gestion des images et le reste du développement.

2.1.5 Référent qualité

Marie est responsable de la qualité du projet. Elle s'assure que le projet respecte les standards, avec le minimum de bugs possible et qu'il soit conforme aux exigences initiales.

2.1.6 Developpeurs

L'équipe de développement est composée de plusieurs membres qui se consacrent à la réalisation technique du projet. Ils sont responsables de l'écriture, des tests et de l'optimisation du code, de la base de données... Chaque développeur s'est vu attribuer un module spécifique à développer. Bien évidemment, cette attribution peut évoluer au fil du temps en fonction de l'envie de chacun et des besoins du projet. Leur travail repose sur les directives techniques établies et le respect des bonnes pratiques de développement.

2.2 Réunions et coordination

La coordination de l'équipe a été principalement assurée durant les créneaux dédiés dans l'emploi du temps. Ces moments réservés ont permis de faire régulièrement le point sur l'avancement du projet, d'échanger sur les tâches en cours et de résoudre les éventuelles difficultés

rencontrées. En dehors de ceux-ci, nous utilisons Discord pour communiquer et transmettre des informations.

Durant la 9ème semaine du projet, une réunion avec notre encadrant a été organisée. Cette rencontre a permis à l'ensemble de l'équipe de partager les avancées réalisées, de clarifier les besoins spécifiques pour la suite du développement.

Pour suivre efficacement la progression des différentes tâches, nous avons utilisé l'outil de gestion de projet *OpenProject* mis à notre disposition. Cet outil a permis de structurer les activités, d'attribuer des responsabilités à chaque membre et d'assurer une visibilité constante sur l'état d'avancement global du projet.

2.3 Évolution de l'organisation

Au cours du projet, peu de changements ont été apportés à l'organisation initiale de l'équipe. Chaque membre a globalement conservé son rôle défini au début du projet.

Cependant, un ajustement en fin de projet a été effectué concernant le groupe chargé de la base de données. Celui-ci est venu apporter son aide à l'équipe web notamment pour ce qui concerne la partie recherche du site web.

3 Répartition effective du travail

3.1 Tâches initialement prévues

Avant le début du projet, un certain nombre de tâches avaient été définies pour assurer le bon déroulement du projet. Ces tâches étaient organisées autour de la conception de la plateforme web, de la gestion des utilisateurs, du traitement des images pour générer des patrons, ainsi que de l'infrastructure nécessaire pour soutenir le projet. La répartition des tâches s'est basée sur les compétences de chaque membre de l'équipe et a été associée à chaque groupe constitué (Image, Web, Data, DevOps).

3.1.1 Conception de la plateforme web

L'un des objectifs était de créer une interface web claire et intuitive, permettant une expérience utilisateur optimale. Les tâches associées étaient les suivantes :

- **Création de l'interface utilisateur :**
 - **Conception de la page d'accueil :** Affichage des informations principales du projet, présentation du concept, et accès à la plateforme de création de patrons.
 - **Page de profil :** Permet à l'utilisateur de gérer ses informations personnelles, de décrire son profil et de visualiser ses créations passées.
 - **Page explorer :** Permet à l'utilisateur de découvrir et d'explorer les créations des autres membres de la communauté, avec une fonction de recherche.
 - **Page convertir :** Interface permettant à l'utilisateur de télécharger une image et de la convertir en patron de broderie en utilisant l'algorithme de réduction de couleurs K-means.
- **Gestion des utilisateurs :**
 - Mise en place d'un système d'inscription avec l'email, le pseudo et le mot de passe.
 - Authentification des utilisateurs via l'email et le mot de passe

3.1.2 Traitement des images pour générer des patrons

Le cœur du projet résidait dans le traitement des images pour les convertir en patrons de broderie. Les tâches prévues dans ce domaine comprenaient :

- **Développement de l'algorithme de réduction des couleurs :**
 - Implémentation de l'algorithme K-means pour la réduction du nombre de couleurs des images tout en préservant les détails nécessaires à la création des patrons.
 - Test d'autres algorithmes comme K-médiane et la ligne de partage des eaux pour comparer les résultats obtenus.
- **Pré-traitement des images :**
 - Application de techniques de dilatation et d'érosion pour éliminer le bruit des images avant leur traitement par K-means.
- **Post-traitement des résultats :**
 - Construction du patron à partir des clusters générés, en assignant une couleur spécifique à chaque zone du patron.
 - Calcul de la longueur de fil nécessaire pour chaque couleur, basé sur la taille du patron et les données de conversion des couleurs.
 - Intégration du nuancier DMC pour associer les couleurs générées aux fils de broderie spécifiques.

- Redimensionnement des images pour les adapter à une grille de taille uniforme (par exemple, 8x8) avant leur transformation en patron.

3.1.3 Gestion des fichiers et des données

L'une des tâches était de garantir une gestion des images, ainsi qu'une base de données bien structurée. Les tâches suivantes étaient prévues :

- **Stockage des fichiers :**
 - Stockage des images sur le serveur avec une organisation permettant un accès rapide.
 - Sauvegarde du chemin d'accès des images dans la base de données pour faciliter leur récupération lors de l'affichage des patrons.
- **Gestion de la base de données :**
 - Création d'une base de données relationnelle (PostgreSQL) pour stocker les informations des utilisateurs, les patrons créés et l'image originale ainsi que les informations associées (date de création, description, etc.).
 - Élaboration de requêtes pour récupérer les données des utilisateurs et des patrons générés.

3.1.4 Infrastructure technique et DevOps

L'objectif était de garantir une infrastructure permettant d'accéder à notre site depuis l'extérieur sans intervention de l'utilisateur.

- **Développement backend :**
 - Création d'une API RESTful en Node.js pour gérer les requêtes des utilisateurs, interagir avec la base de données, et coordonner le traitement des images.
- **Conteneurisation des services :**
 - Utilisation de Docker pour conteneuriser l'application, la base de données, et les services de traitement des images afin d'assurer la portabilité et de simplifier les déploiements.
 - Mise en place de Docker Compose pour orchestrer les interactions entre les différents services dans un environnement unifié.
- **Configuration et gestion d'Apache :**
 - Mise en place d'un serveur web Apache pour gérer les requêtes HTTP entrantes et assurer la redirection vers les bons services.
 - Modification des fichiers de configuration Apache (comme le fichier 'httpd.conf' ou les fichiers de configuration des hôtes virtuels) pour configurer des proxys inverses pointant vers les conteneurs backend.
- **Automatisation des déploiements :**
 - Utiliser des scripts pour pull automatiquement le projet depuis le git ainsi que redémarrer tous les services nécessaires pour mettre à jour l'application.

3.2 Tâches effectivement réalisées

Dans le cadre de ce projet, la majorité des fonctionnalités prévues ont été développées et intégrées. Cependant, certaines tâches restent encore à finaliser avant le rendu. Voici un état des lieux des réalisations et des travaux restants :

Tâches réalisées

- Développement d'un site web fonctionnel.
- Développement et intégration de l'algorithme **K-means** pour la réduction des couleurs.
- Création et gestion des matrices pour la génération des patrons au format 8x8.
- Mise en place des traitements d'image préliminaires, incluant la dilatation et l'érosion.
- Développement de l'interface web.
- Conception de la base de données pour stocker les utilisateurs, les patrons et les métadonnées associées.
- Construction de l'API pour gérer les échanges entre le frontend et le backend (PostgreSQL et Python).

Tâches restantes

- **Gestion de l'oubli de mot de passe** : Cette tâche n'a pas été prise en compte à ce stade.
- **Calcul de la longueur des fils nécessaires** : Prévu initialement, ce calcul sera réalisé si le temps le permet.
- **Achever la page de gestion des comptes, incluant :**
 - L'affichage et l'édition des patrons d'un utilisateur.
 - La personnalisation du profil.
- **Barre de recherche** : Implémenter la gestion de la barre de recherche.

Dans l'ensemble, les tâches réalisées couvrent les fonctionnalités principales du projet. Le temps restant devrait nous permettre de développer les éléments restants dans le temps imparti.

3.3 Analyse des écarts

La majorité des écarts observés entre le plan initial et le travail réalisé s'expliquent principalement par un manque de temps. Contrairement à ce que nous aurions pu anticiper, nous n'avons pas rencontré de problèmes techniques majeurs qui auraient pu bloquer l'avancement du projet. Les ajustements apportés en cours de route ont été réalisés dans le but d'optimiser les ressources disponibles et d'améliorer l'expérience utilisateur.

4 Fonctionnalités développées

4.1 Le développement du site web

Notre site web jouera le rôle d'interface entre l'utilisateur et les différentes fonctionnalités proposées.

Pour structurer ces fonctionnalités, une arborescence a été mise en place :

- **Page d'accueil** : Présentation générale du site avec des explications sur son fonctionnement et des liens vers les principales fonctionnalités.
- **Page d'importation** : Permet aux utilisateurs d'importer une image, de configurer les paramètres nécessaires pour générer un patron, de visualiser le résultat et de le télécharger.
- **Page d'exploration** : Offre la possibilité d'explorer les patrons générés par d'autres utilisateurs, de les noter et de les rechercher via des filtres.
- **Page de profil** : Présente les informations de l'utilisateur, ainsi que les patrons qu'il a générés.

L'ensemble des pages dispose d'une barre de navigation commune permettant aux utilisateurs de naviguer facilement entre les différentes sections du site. De plus, une fonctionnalité de connexion et d'inscription est accessible sur toutes les pages.

4.1.1 Justification des choix

Pour la création du site nous avons fait le choix d'utiliser html/css/js pour le frontend client et Node.js coté serveur. Ce choix repose sur plusieurs critères. Node.js, grâce à son architecture événementielle et non bloquante, est particulièrement adapté pour la gestion des applications web. De plus, l'écosystème riche de modules disponibles via npm facilite l'intégration de bibliothèques et outils nécessaires à notre projet. De plus, Joris avait déjà utilisé cet outil lors de projet précédent.

Le rôle du serveur Node.js

Dans notre projet, le serveur Node.js joue un rôle central en tant que moteur principal du backend. Son rôle peut être décomposé en plusieurs points clés :

1. Gestion des requêtes HTTP : Le serveur Node.js est responsable de la réception et du traitement des requêtes HTTP provenant du frontend. Cela inclut des opérations comme l'importation d'images, la configuration des paramètres pour la génération des patrons et la récupération des patrons stockés dans la base de données. Grâce à des frameworks comme Express.

2. Communication avec les services externes : Le serveur agit comme un intermédiaire entre le frontend et les services externes. Par exemple : - L'envoi des images au script Python pour l'algorithme de génération et de réduction des couleurs. - La communication avec la base de données PostgreSQL pour stocker ou récupérer les patrons générés et les informations utilisateur. - La gestion des authentifications (inscriptions et connexions utilisateur).

3. Traitement des données : Le serveur Node.js est également responsable de la validation et de la transformation des données avant leur stockage ou leur utilisation. Par exemple, il peut vérifier la validité des paramètres reçus (taille de la grille, nombre de couleurs, etc.) et s'assurer que les données envoyées au script Python sont conformes aux attentes.

4. Sécurisation et gestion des erreurs : Le serveur s'assure également que les données utilisateurs sont protégées en appliquant des bonnes pratiques de sécurité (hashage des mots de passe, protection contre les injections SQL, etc.). De plus, il gère les erreurs (comme une image non conforme ou un paramètre manquant) pour offrir des réponses claires au frontend et éviter tout plantage de l'application.

L'organisation frontend

La structure du projet suit une organisation logique pour simplifier le développement et la maintenance :

- **Répertoire views** : Ce dossier contient des fichiers au format `.ejs` (Embedded JavaScript), permettant l'utilisation de templates communs sur les différentes pages de l'application. Par exemple, la barre de navigation (`navbar`) est définie dans un template partagé, garantissant une cohérence visuelle et fonctionnelle entre les pages et évitant la duplication de code.
- **Répertoire css** : Un fichier CSS commun est utilisé pour styliser l'ensemble de l'application. Cette centralisation du style facilite les modifications et assure une uniformité graphique.
- **Fichiers JavaScript spécifiques** : Chaque page de l'application dispose de son propre fichier `.js`, permettant une séparation claire des responsabilités. En complément, des fichiers JavaScript de bibliothèques tierces sont également inclus pour enrichir les fonctionnalités interactives.
- **Fichier index.js** : Ce script représente le serveur principal de l'application. Il gère les routes, les connexions au backend, ainsi que la logique principale de l'application.
- **Fichiers de configuration** : Le projet inclut des fichiers `package.json` et `package-lock.json` pour gérer les dépendances de l'application. Un fichier `config.json` est également utilisé pour centraliser les paramètres configurables du projet, comme les informations de connexion à la base de données, les ports, ou les chemins d'accès aux données.

Cette structure permet une séparation entre le front-end, le back-end, et les ressources partagées, facilitant l'évolution du projet.

4.1.2 Importation d'images et visualisation (WEB)

Descriptif fonctionnel

La fonctionnalité d'importation d'images permet aux utilisateurs de charger une image depuis leur appareil afin de la convertir en un patron de broderie au point de croix. L'interface propose plusieurs options de personnalisation, notamment :

- **Nombre de couleurs** : Contrôle le nombre de couleurs à inclure dans le patron.
- **Dimensions du patron** : Possibilité de définir la taille de la grille en points (par exemple, 100x100).
- **Redimensionnement** : Conserver les proportions de l'image ou l'étendre pour correspondre aux dimensions spécifiées.

Une fois les paramètres définis et l'image importée, un aperçu visuel interactif est proposé. Cet aperçu permet de comparer l'image originale avec une version optimisée contenant des

couleurs réduites grâce au clustering K-means. Le résultat final peut alors être converti et présenté sous la forme d'un patron où chaque point de croix est représenté par une petite figure. Ce patron peut alors être téléchargé par l'utilisateur. En parallèle, si l'utilisateur est connecté, l'image et le patron seront stockés sur le serveur et les informations associées dans la base de données

Descriptif technique

L'implémentation de cette fonctionnalité repose sur plusieurs éléments techniques :

- **Interface utilisateur** : Réalisée en HTML et CSS les interactions avec les paramètres sont gérées via JavaScript.
- **Traitement des images** : Les images importées sont transmises au serveur via des appels API. Elles sont ensuite traitées par un script Python utilisant l'algorithme K-means pour réduire le nombre de couleurs.
- **Retour des résultats** : Une fois le traitement effectué, le serveur retourne un code de retour qui indique que tout s'est bien déroulé ou non. Il nous suffit alors d'aller chercher l'image grâce au chemin donné à l'api et affiché le résultat à l'utilisateur. Suite à une demande de l'utilisateur celui-ci peut aussi être affiché sous la forme de patron en dessous de la prévisualisation et être téléchargé.

Découpage en tâches

Le développement de cette fonctionnalité a été décomposé en plusieurs étapes :

1. Conception de l'interface utilisateur pour l'importation d'images et la sélection des paramètres.
2. Implémentation du backend pour recevoir et traiter les images.
3. Développement de l'aperçu interactif pour afficher les comparaisons avant/après.
4. Gestion des erreurs et validation des fichiers importés (format, taille, etc.).

4.1.3 Recherche de patrons (WEB)

Descriptif fonctionnel

La fonctionnalité de recherche de patrons permet aux utilisateurs de parcourir les créations disponibles sur la plateforme. Chaque patron est associé à des attributs tels que des **tags**, une **date de création** et une **note / score de popularité** (Encore en cours de réflexion). Les utilisateurs peuvent effectuer une recherche par mots-clés grâce aux tags associés à chaque patron.

En plus de la recherche, des filtres permettent de trier les résultats selon plusieurs critères. Il est possible d'afficher les patrons les plus récents en sélectionnant une période spécifique (dernier jour, dernière semaine ou dernier mois). Les utilisateurs peuvent également trier les créations par popularité, celle-ci étant déterminée par un système de votes où chaque utilisateur peut recommander ou descendre un patron via des boutons dédiés (système up/down).

Enfin, chaque image affichée dans les résultats est interactive. En cliquant sur une image, l'utilisateur peut voter pour modifier le score de popularité, télécharger le patron correspondant ou accéder à des informations supplémentaires, telles que le créateur de l'image.

Descriptif technique

Techniquement, cette fonctionnalité repose sur la **base de données** où chaque patron est associé à des attributs tels que des tags pour la recherche, une date de création pour le tri temporel et un score de popularité initialisé à 0 et modifié en fonction des votes des utilisateurs. Le traitement des requêtes de recherche et des filtres est effectué côté serveur via des **appels API**, qui interrogent la base de données et renvoient les résultats au client en ajoutant des clauses pour répondre aux différentes contraintes imposées par les filtres.

L'interface utilisateur est conçue en **HTML/CSS/JavaScript** pour gérer l'affichage et les interactions dynamiques. Les images affichées sont présentées sous forme de vignettes interactives, permettant de voter, télécharger le patron ou afficher des détails supplémentaires. Les résultats de recherche sont affichés en temps réel.

Découpage en tâches

Le développement de cette fonctionnalité a été découpé en plusieurs étapes :

- **Modélisation de la base de données** : La première étape a consisté à modéliser la base de données pour inclure les attributs nécessaires tels que les tags, la date de création et le score de popularité.
- **Développement de l'API** : Ensuite, une API a été développée pour gérer les requêtes de recherche et de filtrage des patrons.
- **Mise en place de l'interface utilisateur** : L'interface utilisateur a été mise en place, comprenant la création de la barre de recherche, des filtres et des vignettes interactives permettant d'afficher les résultats de recherche.
- **Gestion des interactions utilisateur** : La gestion des interactions utilisateur a été ajoutée, notamment pour le système de votes et le téléchargement des patrons générés.
- **Optimisation des performances** : La gestion des performances a été intégrée pour garantir un affichage rapide des résultats, même pour une base de données volumineuse.

4.1.4 Connexion et page de profil (WEB)

Descriptif fonctionnel

La fonctionnalité de connexion et de gestion de profil permet aux utilisateurs d'accéder à leur espace personnel sur la plateforme. Lors de leur première utilisation, les utilisateurs peuvent s'inscrire en fournissant une adresse mail, un pseudo, et un mot de passe. Une fois inscrits, ils peuvent se connecter en saisissant leur adresse mail et leur mot de passe.

L'authentification est gérée via un système de **token JWT** qui est stocké soit dans le **sessionStorage** soit dans le **localStorage**, selon le choix de l'utilisateur. Ce choix détermine si la session reste active après la fermeture du navigateur (*rester connecté*).

Une fois connectés, les utilisateurs ont accès aux fonctionnalités principales, notamment la conversion d'images en patrons de broderie et l'accès à leur page de profil. Sur cette page, ils peuvent :

- Modifier leurs **informations personnelles** (pseudo, mot de passe, etc.).
- Ajouter ou éditer une **description personnelle** visible sur leur profil.
- Visualiser leurs **créations précédentes**, avec la possibilité de télécharger ou de supprimer des patrons.

Enfin, un bouton dédié permet de se déconnecter de la plateforme en supprimant le token **JWT** du stockage choisi.

Descriptif technique

Techniquement, cette fonctionnalité repose sur plusieurs éléments clés :

- **Système d'authentification :**
 - Lors de l'inscription, les informations utilisateur (adresse mail, pseudo, mot de passe hashé) sont enregistrées dans la **base de données** en vérifiant leur validité.
 - À la connexion, un **token JWT** est généré et transmis au client. Ce token contient des informations d'authentification et d'autorisation (par exemple, l'ID utilisateur).
 - Le token peut être stocké dans le **sessionStorage** (session temporaire) ou le **localStorage** (session persistante), selon le choix de l'utilisateur.
- **Page de profil :**
 - Les informations utilisateur sont récupérées depuis la base de données à l'aide d'appels API.
 - Les modifications effectuées sur le profil (par exemple, la description ou les informations personnelles) sont transmises au serveur via des requêtes.
 - La liste des créations précédentes est générée dynamiquement à partir des données associées à l'utilisateur.
- **Déconnexion :**
 - Lorsqu'un utilisateur se déconnecte, le token JWT est supprimé du *storage*, invalidant la session côté client.

% Exemple de gestion des tokens en JavaScript :

```
1 if (stayConnected) {  
2     localStorage.setItem('authToken', token);  
3 } else {  
4     sessionStorage.setItem('authToken', token);  
5 }
```

Découpage en tâches

Le développement de cette fonctionnalité a été découpé en plusieurs étapes :

1. Mise en place du système d'inscription avec vérification des champs (adresse mail valide, mot de passe sécurisé, etc.).
2. Développement du système de connexion avec génération et stockage des tokens JWT.
3. Création de la page de profil avec des fonctionnalités permettant :
 - La modification des informations personnelles et de la description.
 - La visualisation des créations précédentes.
4. Implémentation du système de déconnexion (suppression du token JWT).
5. Ajout de vérifications côté serveur pour sécuriser les API (vérification des tokens JWT, contrôle des permissions).
6. Tests utilisateurs pour valider les interactions (connexion, navigation, déconnexion, etc.).

4.2 Conversion d'image en patron (SCRIPT)

4.2.1 Descriptif fonctionnel

La conversion d'une image en patron constitue une fonctionnalité clé de notre projet. Elle transforme une image en un motif optimisé pour le point de croix, reposant sur plusieurs étapes :

- **Réduction des couleurs** : Utilisation de l'algorithme **K-means** pour simplifier la palette de couleurs.
- **Traitements préliminaires** : Nettoyage de l'image via des opérations de dilatation et d'érosion.
- **Redimensionnement de l'image** : Transformation de l'image en une grille 8x8 pour aligner chaque pixel sur une case de la grille.
- **Optimisation des clusters** : Identification du nombre optimal de couleurs à l'aide de la méthode **Elbow**.
- **Validation des résultats** : Évaluation des performances par le calcul de l'**erreur moyenne quadratique (MSE)** entre le patron et l'image originale.

Pour l'utilisateur final, cette fonctionnalité aboutit à un patron associant à chaque symbole une couleur du nuancier DMC.

4.2.2 Répartition du travail

Le travail a été organisé en plusieurs phases distinctes :

- **Recherche et documentation** : Tous les membres ont participé à une première phase d'exploration des techniques possibles, y compris l'étude de plusieurs algorithmes tels que K-Means, K-Médiane, et la ligne de partage des eaux.
- **Développement du K-means** : Malek et Johann ont implémenté l'algorithme, optimisé les clusters et intégré les comparaisons nécessaires pour assurer des résultats cohérents.
- **Traitements pré et post-algorithme** : Marie et Tristan ont développé les méthodes de dilatation, d'érosion et de redimensionnement.
- **Construction des matrices finales** : Tristan, Malek et Marie ont collaboré à la construction des patrons en utilisant des grilles 8x8.
- **Débogage** : L'ensemble de l'équipe a corrigé les erreurs et validé les résultats via des tests, une phase qui a requis environ 14 heures.

4.2.3 Justification techniques - Clustering avec K-means

Le choix de l'algorithme K-means pour la réduction des couleurs a été guidé par plusieurs considérations techniques et pratiques.

Parmi les options explorées, telles que les algorithmes K-médiane et la ligne de partage des eaux, K-means s'est démarqué par son équilibre entre la qualité des résultats et sa rapidité d'exécution. Contrairement à d'autres approches, il permet de réduire efficacement le nombre de couleurs tout en préservant la structure visuelle de l'image. Ces caractéristiques le rendent particulièrement adapté à la génération de patrons de broderie, où chaque couleur doit correspondre à une nuance précise du nuancier DMC.

L'implémentation de K-means a été réalisée en Python, en utilisant des bibliothèques comme **scikit-learn** pour garantir une intégration avec les autres composants du projet. Les appels API entre le serveur Node.js et les scripts Python facilitent le traitement des images en temps réel, renforçant ainsi la cohérence ainsi que la séparation des tâches dans le groupe dans le développement du projet.

4.2.4 Fonctionnalités développées et à venir

Fonctionnalités développées :

- Mise en place d'un pipeline complet de conversion d'images utilisant K-means, incluant les étapes de nettoyage, redimensionnement, et validation.
- Optimisation des clusters via la méthode **Elbow**.

- Validation des patrons générés grâce au calcul de la **MSE**.

Fonctionnalités à venir :

- Calcul de la longueur de fil nécessaire pour chaque couleur du nuancier DMC.

4.2.5 Techniques utilisées

Les principales techniques employées incluent :

- **Dilatation et érosion** : Élimination des artefacts visuels et réduction du bruit dans l'image d'entrée.
- **Réduction des couleurs** : Utilisation de **K-means** pour compresser la palette, avec ajustement du nombre de clusters par la méthode **Elbow**.
- **Validation** : Calcul de l'erreur moyenne quadratique (**MSE**) pour identifier le patron le plus fidèle à l'image d'origine.

4.2.6 Problèmes rencontrés

Au cours du développement, plusieurs difficultés ont été identifiées et résolues :

- **Gestion des centroides** : Certaines couleurs manquaient lorsque les centroides n'étaient pas correctement utilisés pour associer les couleurs. Ce problème a été corrigé en ajustant l'affectation des couleurs aux clusters.
- **Redimensionnement après K-means** : Une erreur dans l'ordre des opérations introduisait de nouvelles couleurs lors du redimensionnement de l'image. La solution a consisté à effectuer le redimensionnement avant K-means.

4.2.7 Apprentissages

Ce projet a permis à tous les membres de l'équipe de se familiariser avec des concepts et des outils importants, notamment :

- La compréhension approfondie de l'algorithme **K-means** et de ses variantes.
- Les techniques de traitement d'image comme la dilatation, l'érosion, et la gestion des couleurs.
- La méthodologie de validation des résultats via le calcul de la MSE.

4.2.8 Exemples de code

```

1 # Methode Elbow pour determiner le nombre optimal de clusters
2 def deterK(img):
3     X = img.reshape((-1, 3))
4     X = pd.DataFrame(X)
5
6     inertia_list = []
7     k_list = range(1, 10)
8
9     # Calcul de l'inertie pour chaque K
10    for k in k_list:
11        kmeans = KMeans(n_clusters=k)
12        kmeans.fit(X)
13        inertia_list.append(kmeans.inertia_)
14
15    kl = KneeLocator(range(0, 9), inertia_list, curve="convex", direction="decreasing")
16    k_optimal = kl.elbow

```

```

17     print("Le nombre optimal de clusters est : ", k_optimal)
18     return k_optimal

1 # Calcul de l'erreur moyenne quadratique (MSE) pour valider un patron
2 def CalculMSE(imgATraite, imgOrigine, tentative, K=None):
3     results = []
4     errors = []
5     for i in range(tentative):
6         imgTransformed, center = Kmeans(imgATraite, K)
7         results.append(imgTransformed)
8
9     # Calcul de la MSE entre l'image originale et l'image transformee
10    error = mse(imgOrigine, imgTransformed)
11    errors.append(error)
12    print(f"Erreur pour la transformation {i+1} : {error}")
13    return errors, results, center

```

4.2.9 Exemples sur des images



FIGURE 4.1 – Image originale



(a) K-means avec $k = 3$



(b) K-means avec $k = 6$



(c) K-means avec $k = 9$



(d) K-means avec $k = 12$

FIGURE 4.2 – Utilisation de k-means

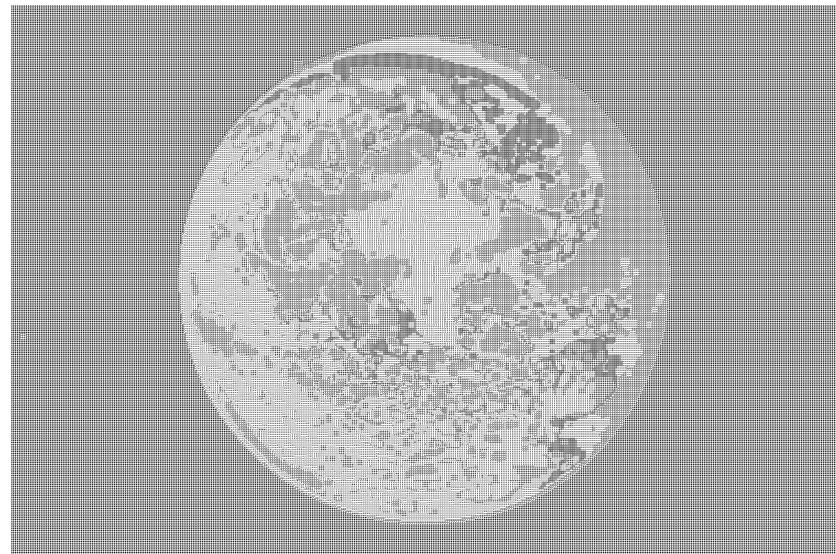


FIGURE 4.3 – Patron de broderie pour $k = 12$

Pour d'autres exemples, vous pouvez utiliser notre [site](#) et faire vos propres tests (sur le réseau de la fac ou en utilisant son VPN).

4.3 Stockage et gestion des patrons

4.3.1 Descriptif fonctionnel

La base de données est organisée pour gérer les utilisateurs, les patrons et l'image originale ainsi que les métadonnées associées. Voici les fonctionnalités principales :

- **Gestion des utilisateurs** : Chaque utilisateur est identifié via une clé unique et possède des informations comme son nom, son email, son mot de passe et sa date de création.
- **Stockage des images** : Les images originales utilisées par les utilisateurs sont sauvegardées dans une table dédiée. Chaque image est associée à son propriétaire et possède des informations comme son chemin de stockage et de sa date d'importation.
- **Création et gestion des patrons** : Chaque patron est généré à partir d'une image et contient des informations supplémentaires comme une description, un nombre estimé de fils nécessaires, des tags pour faciliter les recherches, et son chemin de stockage.
- **Noter les patrons** : Laisser la possibilité aux utilisateurs de noter les patrons
- **Association des couleurs** : Les couleurs utilisées dans les patrons sont stockées dans une table dédiée avec leurs valeurs RGB et leur équivalent DMC. Cela permet de gérer la correspondance entre les patrons générés et les fils de broderie.

4.3.2 Choix techniques

Pour ce projet, l'équipe a choisi d'utiliser PostgreSQL, un système de gestion de bases de données relationnelles. L'objectif était d'apprendre à utiliser un nouvel outil, car aucun des membres n'avait encore travaillé avec PostgreSQL. Ce choix a permis de découvrir ses fonctionnalités avancées et sa flexibilité. L'implémentation a nécessité l'utilisation de requêtes SQL avancées, notamment pour gérer les recherches.

4.3.3 Descriptif technique

Structure de la base de données :

- **Table Utilisateur** : Contient les données personnelles des utilisateurs et leurs informations de connexion.
- **Table Image** : Stocke les images importées par les utilisateurs avec des métadonnées telles que le chemin d'accès, la date d'importation et une note.
- **Table Patron** : Relie les images aux patrons générés, incluant des informations sur la description, les tags, et la consommation estimée de fils.
- **Table Note** : Permet de stocker sans redondance de données les utilisateurs qui ont notés un patron ainsi que la note.
- **Table Couleur** : Gère les couleurs utilisées dans les patrons, associant leurs codes RGB aux références DMC pour garantir une correspondance avec les fils de broderie.

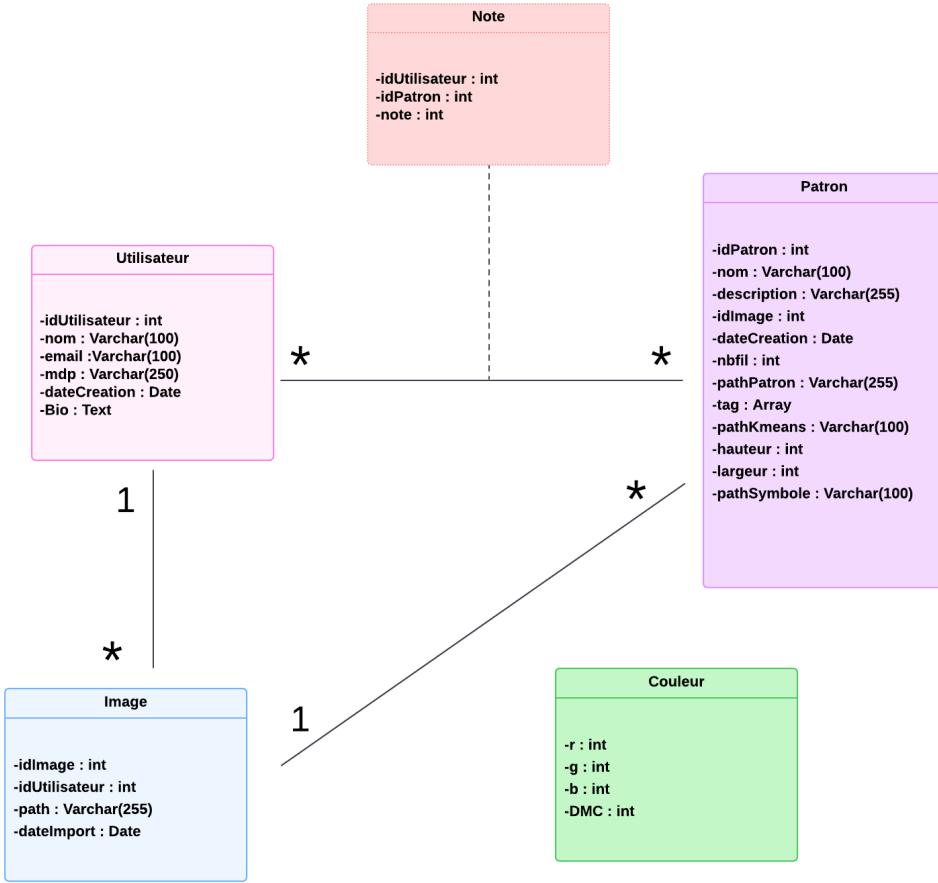


FIGURE 4.4 – Diagramme base de données

Technologies utilisées :

- **PostgreSQL** : Choisi pour ses performances et la possibilité de répondre à nos besoins relationnels.
- **Node PostgreSQL Module** : Pour connecter l'application backend développée en Node.js à la base de données PostgreSQL, le projet utilise le module `pg`, qui est l'un des modules les plus populaires pour interagir avec PostgreSQL depuis un serveur Node.js.

4.3.4 Découpage en tâches

Le travail sur la base de données a été divisé en plusieurs étapes :

- **Modélisation de la base :**
 - Conception des tables et des relations pour répondre aux besoins fonctionnels (Ines, Lysia, Massissilia).
- **Développement des fonctionnalités :**
 - Création des requêtes pour insérer et rechercher des images et des patrons (Ines, Lysia, Massissilia).
 - Implémentation de la gestion des notes pour ajouter des notes aux patrons (Lysia, Ines)

- Développement des filtres et requête pour trier les patrons par notes ou par date (Massissilia, Ines).
- **Tests et intégration :**
 - Validation des requêtes et intégration avec l'application pour garantir la synchronisation des données (Lysia).

4.3.5 Justification des écarts

Des écarts par rapport aux attentes initiales ont été observés :

- **Ajout de nouvelles fonctionnalités :** La possibilité de rechercher des patrons
- **Ajout de nouvelles fonctionnalités :** La possibilité de télécharger les images via un bouton.

4.4 Tâches réalisées par l'équipe DevOps

4.4.1 Installation et configuration du DNS

Le but de cette partie est d'expliquer la mise en œuvre de notre serveur de noms et les choix faits pour sa configuration. L'objectif est de permettre de joindre notre site web via un navigateur internet grâce à l'URL **www.lesbrodeurs.dijon**.

Nous avons commencé par installer BIND9, le paquet nécessaire pour configurer un serveur DNS.

Ensuite, nous avons créé notre domaine **lesbrodeurs.dijon**, avec une zone DNS contenant trois enregistrements principaux :

- **Un enregistrement de type NS :** Désigne notre serveur comme responsable de la gestion des noms pour le domaine, avec les adresses IP 172.31.60.16 (serveur de développement) et 172.31.60.17 (serveur de production).
- **Un enregistrement de type A :** Associe le nom **debian** (nom de notre serveur de développement) à l'adresse IP mentionnée.
- **enregistrement de type CNAME :** Définit un alias **www** pointant vers **debian.lesbrodeurs.dijon**, ce qui permet d'accéder au site via l'URL souhaitée.

Une zone DNS inverse a également été configurée avec les enregistrements suivants :

- **Un enregistrement de type NS :** Désigne le serveur DNS responsable de la zone inverse.
- **Un enregistrement de type PTR :** Associe l'adresse IP 172.31.60.16 ou 172.31.60.17 au nom **debian.lesbrodeurs.dijon**.
- Il n'est pas nécessaire de créer un enregistrement PTR pour un alias (CNAME), car lors d'une recherche DNS inverse, le serveur retourne directement le nom canonique (enregistrement A) auquel le CNAME est lié.

Nous avons testé notre configuration avec la commande **nslookup** et remarqué que notre serveur ne prenait pas en compte notre DNS par défaut. Cela s'explique par la configuration du fichier **NetworkManager**, qui utilisait un autre serveur DNS, celui de l'IEM. Pour résoudre ce problème, nous avons modifié le fichier de configuration du **NetworkManager** pour remplacer le DNS par défaut par l'adresse IP de notre DNS. Cette modification permet de régénérer un fichier **resolv.conf** contenant le bon serveur DNS. Le **NetworkManager** était déjà installé sur notre serveur, car celui-ci est une machine virtuelle préinstallée et configurée avec le DNS de l'IEM comme serveur principal.

À la fin de cette configuration, notre serveur DNS est entièrement fonctionnel. Cependant,

comme le domaine **lesbrodeurs.dijon** n'est connu que de notre DNS, il est accessible uniquement aux machines configurées pour utiliser notre serveur. Pour élargir l'accès il faudrait :

1. Changer la configuration DNS des ordinateurs pour pointer vers notre serveur DNS.
2. Référencer notre DNS auprès d'un DNS de niveau supérieur (comme celui de l'IEM) pour une plus grande visibilité.

4.4.2 Installation et configuration du serveur web Apache

L'objectif est maintenant de faire en sorte que lorsqu'un utilisateur entre l'URL `www.lesbrodeurs.dijon` dans son navigateur, la page de notre site s'affiche. Comme cette requête est de type HTTP, il nous fallait un serveur web pour gérer ce type de demande.

Nous avons choisi d'utiliser Apache, l'un des serveurs web les plus répandus, car il est fiable, durable et largement supporté. De plus, nous disposons déjà des connaissances nécessaires pour son installation et sa configuration.

Nous avons installé la version 2.4.62 d'Apache pour Debian, qui est la dernière version stable disponible. Une fois l'installation terminée, pour vérifier que tout s'est bien passé et que le port 80 (nécessaire pour les requêtes HTTP) est ouvert, il suffit d'entrer l'URL suivante dans un navigateur :`http://lesbrodeurs.dijon`. Si Apache est correctement installé, la page par défaut d'Apache, située dans `/var/www/html`, s'affichera.

La première étape dans la configuration d'Apache consiste à lui donner l'accès au répertoire où se trouve le code de notre site web. Pour cela, nous avons modifié le fichier `apache2.conf`. Concernant l'emplacement de notre site, nous avons choisi de le placer dans le répertoire `/var/wwwlesbrodeurs.dijon/broderie`(nom du dossier GitLab). Par convention, en administration système, les sites web sont souvent placés dans le répertoire `/var`, car ce sont des entités souvent mises à jour et donc variables. Il aurait également été possible de le stocker dans `/srv`, un autre répertoire conventionnellement utilisé pour les services fournis par le serveur.

Pour finaliser la configuration de notre serveur web, nous avons créé un hôte virtuel pour notre site. Cet hôte est configuré pour écouter sur le port 80, afin de gérer uniquement les requêtes HTTP. Pour que l'accès à notre site se fasse via l'URL `www.lesbrodeurs.dijon`, nous avons spécifié l'alias de notre serveur, configuré précédemment dans notre DNS. Enfin, nous avons attribué à Apache les droits nécessaires pour accéder au répertoire où le code de notre site est stocké sur le serveur.

Il est important de noter que le serveur web et le DNS sont étroitement liés dans cette configuration. En effet, lorsqu'un utilisateur effectue une requête HTTP, son navigateur tente de contacter une machine en utilisant le nom de domaine saisi. C'est pourquoi il est crucial de bien référencer cet alias dans notre DNS. Une fois le serveur web configuré avec cet alias, il peut se reconnaître et répondre correctement à la requête HTTP. Pour vérifier que la configuration a bien été réalisée, nous avons redémarré Apache et entré l'URL `http://www.lesbrodeurs.dijon` dans un navigateur pour voir si la page de notre site web s'afficher correctement.

Notre serveur web Apache est maintenant fonctionnel pour héberger notre site web. Cependant, une amélioration possible consiste à passer à des requêtes HTTPS. Actuellement, les échanges de données entre le client et notre serveur se font en HTTP, ce qui signifie que ces données circulent en clair et peuvent être interceptées. Cela constitue une faille de sécurité, et l'utilisation de HTTPS pour sécuriser ces échanges est une possibilité d'amélioration pour le second semestre.

4.4.3 Installation et configuration du serveur Node.js

En développement web, il existe une distinction entre le code s'exécutant côté client (comme le HTML, CSS, JavaScript) et celui s'exécutant côté serveur (comme le PHP ou JavaScript côté serveur avec Node.js). Le serveur, qui prend en charge une partie de l'exécution du code, nécessite l'installation des langages ou environnements nécessaires à cette tâche.

Dans notre cas, l'équipe de développement web a demandé à l'équipe DevOps d'installer Node.js sur le serveur. Cette partie détaille l'installation et la configuration d'un serveur Node.js.

Node.js est un environnement d'exécution JavaScript côté serveur, souvent utilisé pour gérer des applications web et des API. Il peut répondre à des requêtes HTTP sur un port spécifique, dans notre cas le port 8080. Nous avons choisi d'utiliser Node.js conjointement avec un serveur web Apache, qui réceptionnera les requêtes HTTP sur le port 80 avant de les rediriger vers Node.js via le port 8080. Apache agit ici comme un **reverse proxy**, permettant de simplifier la gestion des requêtes et d'assurer une meilleure compatibilité avec des fonctionnalités avancées.

Une alternative aurait été d'utiliser uniquement Node.js comme serveur principal, car il peut gérer des requêtes HTTP de manière autonome. Cependant, nous avons décidé de conserver Apache comme façade pour plusieurs raisons, notamment pour faciliter l'ajout futur de la prise en charge du protocole HTTPS. Contrairement à Apache, Node.js n'est pas conçu comme un serveur web complet et ne gère pas les requêtes HTTPS de manière native, ce qui complique la mise en place d'une connexion sécurisée.

Concernant l'installation de Node.js, nous avons opté pour la version 20.18.0, qui était à jour au moment de la configuration. La configuration de l'application, notamment l'écoute sur un port, est réalisée à l'aide d'un fichier `index.js` fourni par l'équipe de développement. Ce fichier initialise le processus Node.js nécessaire au bon fonctionnement du site web.

Un problème important s'est posé : que faire en cas d'arrêt du processus Node.js ? Étant donné que ce processus est essentiel au site, son arrêt entraînerait une panne des fonctionnalités dynamiques, ce qui rendrait le site partiellement ou totalement inaccessible. Pour résoudre ce problème, nous avons mis en place un **processus système** sous forme de **démon**. Ce démon, similaire à un service comme httpd pour Apache. Il surveille l'état du processus Node.js et en cas d'arrêt, il relance automatiquement le processus. Il gère également le redémarrage automatique de Node.js lors du redémarrage du serveur. Pour la mise en place de ce démon, nous avons créé un script système dédié. Ce script est placé dans le répertoire `/etc/scripts`.

```

scripts >-- dom.json
1  #!/bin/bash
2
3  # Charger nvm
4  export NVM_DIR="/root/.nvm"
5  [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
6
7  # Vérifie si nvm a été chargé correctement
8  if [ -z "$NVM_DIR" ]; then
9      echo "NVM_DIR non défini. Vérifiez l'installation de nvm."
10     exit 1
11 fi
12
13 # Change le répertoire de travail
14 cd /var/www/wwwlesbrodeurs/broderie || exit
15
16 # Démarrer l'application Node.js
17 node . >> /var/log/nodejs/NodejsOutput.log 2>> /var/log/nodejs/NodejsError.log
18

```

FIGURE 4.5 – Script démarage Node.js

Ce script a pour objectif de charger l'environnement de gestion de Node.js (nvm) pour éviter des erreurs d'exécution, de se placer dans le répertoire où se trouve `index.js`, puis de démarrer le serveur Node.js. Une gestion des logs a également été mise en place pour enregistrer les événements liés au processus Node.js.

```

NodejsError.log ×
var > log > nodejs >  NodejsError.log
1  Error: ENOENT: no such file or directory, stat '/var/www/wwwlesbrodeurs/broderie/html/accueil.html'
2  Error: ENOENT: no such file or directory, stat '/var/www/wwwlesbrodeurs/broderie/html/accueil.html'
3  Error: ENOENT: no such file or directory, stat '/var/www/wwwlesbrodeurs/broderie/html/accueil.html'
4  |

```

FIGURE 4.6 – Fichier de log d'erreur

```

1 Server started on http://localhost:8080
2 Server started on http://localhost:8080
3 Requête reçue pour la connexion
4 Corps de la requête: { email: 'jorisbaud@gmail.com', password: '330b890b793b90341f18f3af4b173947' }
5 Mot de passe incorrect
6 Requête reçue pour la connexion
7 Corps de la requête: { email: 'jorisbaud@gmail.com', password: 'e9837d47b610ee29399831f917791a44' }
8 Utilisateur trouvé:
9 idutilisateur: 11,
10 nom: 'Baud Joris',
11 email: 'joris_baud@gmail.com',
12 mot_de_passe: 'e9837d47b610ee29399831f917791a44',
13 datecreation: 2024-10-14T08:44:09.785Z
14 }
15 Requête reçue pour obtenir les informations de l'utilisateur
16 Email de l'utilisateur: jorisbaud@gmail.com
17 Résultat de la requête:
18   idutilisateur: 11,
19   nom: 'Baud Joris',
20   email: 'joris_baud@gmail.com',
21   datecreation: 2024-10-14T08:44:09.785Z
22 }


```

FIGURE 4.7 – Fichier de log sortie standard

Nous avons ensuite créé un démon `node.service`. Pour cela, un fichier nommé `node.service` a été ajouté dans le répertoire `/etc/systemd/system`, avec la configuration suivante :

```
etc > systemd > system > node.service
1 [Unit]
2 # Description de votre service
3 Description= NodeJS DAEMON
4
5 # Pour démarrer après le réseau actif sur votre serveur
6 After=network.target
7
8 [Service]
9 # Votre répertoire de travail, afin d'avoir des chemins relatifs cohérents
10 WorkingDirectory=/var/www/wwwlesbrodeurs/broderie
11
12 # Le fichier applicatif à lancer (ce dernier est relatif à la directive WorkingDirectory)
13 ExecStart=/etc/scripts/dem_js.sh
14
15 # Si le service crash, il essaie de le relancer
16 Restart=on-failure
17
18 # Timeout de 500ms entre le crash et le redémarrage
19 RestartSec=500ms
20
21 # On envoie les logs vers le syslog
22 StandardOutput=syslog
23 StandardError=syslog
24
25 # Nom du processus dans le syslog
26 SyslogIdentifier=nodejs
27
28 # Utilisateur lançant le service
29 User=root
30
31 [Install]
32 # équivalent au runlevel 3 d'init
33 WantedBy=multi-user.target
```

FIGURE 4.8 – node.service

La journalisation des processus permet le débogage en cas de problème. Elle permet de retracer les événements et d'identifier rapidement les causes potentielles d'une panne ou d'un dysfonctionnement.

Le serveur applicatif Node.js est désormais opérationnel, et la gestion de son activité, notamment via le démon `node.service`, est également assurée permettant la continuité de service.

4.4.4 Installation et configuration de la base de données PostgreSQL

Dans cette section, nous abordons l'installation et la configuration de la base de données. Cette tâche a été réalisée en collaboration avec l'équipe spécialisée en gestion des bases de données, qui a demandé l'installation du SGBD relationnel PostgreSQL.

Nous avons effectué une installation classique de PostgreSQL. L'essentiel du travail s'est ensuite concentré sur la configuration pour répondre aux besoins spécifiques du projet.

Au niveau du SGBD, l'utilisateur `postgres` conserve son rôle de super-utilisateur. Ce rôle permet d'administrer le SGBD, notamment pour créer ou supprimer des bases de données.

Une base de données nommée `les_brodeurs` a été créée pour répondre aux besoins du site web. Afin de garantir une gestion sécurisée et structurée des accès, plusieurs rôles ont été définis pour cette base :

- Trois rôles principaux ont été définis :

1. `admin_les_brodeurs` : Droits étendus pour la création, modification et suppression des tables.
2. `utilisateur_les_brodeurs` : Droits pour effectuer des opérations `INSERT`, `UPDATE`, et `DELETE`.
3. `user_dev_web` : Droits limités pour l'accès via l'application web.

Une attention particulière a été portée à la gestion des droits sur la base de données pour garantir la sécurité des données. L'attribution des rôles et permissions a été un point délicat, nécessitant des ajustements en fonction des retours de l'équipe BDD et des besoins opérationnels.

PostgreSQL a été installé avec succès, et la base de données `les_brodeurs` est opérationnelle.

4.4.5 Installation et configuration de l'environnement python

L'équipe Image a demandé à l'équipe DevOps d'installer Python en version 3 afin de gérer les besoins liés au traitement des images.

Pour l'exécution du code Python, l'équipe Image a décidé de mettre en place un environnement complet basé sur Flask, un framework permettant de développer des applications web ou des API. L'objectif principal pour l'équipe DevOps est maintenant, de déterminer l'emplacement adéquat pour héberger ce serveur Flask. Et de comprendre son fonctionnement en détail pour évaluer s'il nécessite des ajustements au niveau système.

L'équipe Image a fourni un script d'installation de l'environnement Python et des bibliothèques nécessaires au bon fonctionnement de l'API. Les deux équipes ont collaboré pour expliquer son fonctionnement et procéder à la première installation de l'API. Celle-ci génère un processus simple qui écoute sur le port 60001 du serveur. En ce qui concerne le lieu d'installation de l'API, l'équipe DevOps a opté pour le dossier `/srv/wwwlesbrodeurs_services/wwwlesbrodeurs_API/lesbrodeurs-appPython`, conformément à la convention qui désigne ce répertoire pour héberger les services d'un serveur. Un sous-dossier `wwwlesbrodeurs_API/lesbrodeurs-appPython` a été créé à cet effet.

Comme pour le processus Node.js, il est nécessaire de gérer les situations où le processus pourrait s'arrêter de manière inattendue. Ce processus est crucial pour le bon fonctionnement du site, car son arrêt entraînerait une panne des fonctionnalités de génération de patron. Sur le même principe que pour Node.js, un processus système a été mis en place sous forme de démon afin d'assurer sa stabilité.

```
etc > scripts > $ dem_APIpython.sh
1  #!/bin/bash
2
3  # Change le répertoire de travail pour ouvrir l'environnement
4  cd /srv/wwwlesbrodeurs_services/wwwlesbrodeurs_API/lesbrodeurs-appPython || exit
5
6  # Démarrer l'environnement python
7  source bin/activate
8
9  # Change le répertoire de travail pour executer app.py
10 cd /srv/wwwlesbrodeurs_services/wwwlesbrodeurs_API/lesbrodeurs-appPython/Projet || exit
11
12 # Démarrer l'API python
13 python3 app.py >> /var/log/wwwlesbrodeurs_API/APIpythonOutput.log 2>> /var/log/wwwlesbrodeurs_API/APIpythonError.log
14 |
```

FIGURE 4.9 – Script daemon API

Ce script a pour objectif de charger l'environnement python installer avec le script d'installation précédent, de le redémarrer et de relancer l'API. Une gestion des logs a également été mise en place pour enregistrer les événements liés au processus de l'API.

(a) Logs erreurs API python

```
APythonOutput.log
var z = log > wwwlabstreasures_API / APythonOutput.log
1   + Serving Flask app 'app'
2   + Debug mode: off
3   Error pour la transformation 1 : 13583_298647039847
4   Error pour la transformation 2 : 13628_461398476519
5   Error pour la transformation 3 : 13583_298647039847
6   Error pour la transformation 4 : 13583_35988953238
7   Error pour la transformation 5 : 13592_8462875143
8   Error pour la transformation 6 : 13584_877013584762
9   Error pour la transformation 7 : 13592_8462875143
10  Error pour la transformation 8 : 13579_8884
11  Erreur pour la transformation 9 : 13581_57794207152
12  Erreur pour la transformation 10 : 13581_60691394761
13
14 == Associations entre Centroside et Motifs ==
Centroside 1:[151|151|151]
15 Motif associe :
16 [125 255 255 255 255 255 255 255]
17 [125 255 255 255 255 255 255 255]
18 [125 255 255 255 255 255 255 255]
19 [125 255 255 0 0 255 255 255]
20 [125 0 0 0 0 0 255 255]
21 [125 255 255 255 255 255 255 255]
22 [125 255 255 0 0 255 255 255]
23 [125 255 255 0 0 255 255 255]
24 [125 255 255 0 0 255 255 255]
25 [125 255 255 0 0 255 255 255]
26
Centroside 4:[115|115|115]
27 Motif associe :
28 [105 255 255 255 255 255 255 255]
29 [105 255 255 0 0 255 255 255]
30 [105 255 255 0 0 255 255 255]
31 [105 255 0 0 0 255 255]
32 [105 255 0 0 0 255 255]
33 [105 255 0 0 0 255 255]
34 [105 0 0 0 0 0 255]
35 [105 255 255 255 255 255 255 255]
```

(b) Logs infos API python

Nous avons ensuite créé un démon `APIpython_LesBrodeurs.service`. Pour cela, un fichier nommé `APIpython_LesBrodeurs.service` a été ajouté dans le répertoire `/etc/systemd/-system`, avec la configuration suivante :

```
etc > systemd > system > APIpython_LesBrodeurs.service
1 [Unit]
2 # Description de votre service
3 Description= APIpython_LesBrodeurs DAEMON
4
5 # Pour démarrer après le réseau actif sur votre serveur
6 After=network.target
7
8 [Service]
md # Votre répertoire de travail, afin d'avoir des chemins relatifs cohérents
10 WorkingDirectory=/srv/wwwlesbrodeurs_services/wwwlesbrodeurs_API/lesbrodeurs-appPython
11
12 # Le fichier applicatif à lancer (ce dernier est relatif à la directive WorkingDirectory)
13 ExecStart=/etc/scripts/dem_APIpython.sh
14
15 # Si le service crash, il essaie de le relancer
16 Restart=on-failure
17
18 # Timeout de 500ms entre le crash et le redémarrage
19 RestartSec=500ms
20
21 # On envoie les logs vers le syslog
22 StandardOutput=syslog
23 StandardError=syslog
24
25 # Nom du processus dans le syslog
26 SyslogIdentifier=APIpython_LesBrodeurs
27
28 # Utilisateur lançant le service
29 User=root
30
31 [Install]
32 # équivalent au runlevel 3 d'init
33 WantedBy=multi-user.target
```

FIGURE 4.11 – Service API

4.4.6 Migration de serveur de développement vers celui de production

La migration du serveur de développement vers celui de production se déroulera en deux étapes. La première consistera en la préparation du serveur de production, suivie, dans un second temps, de la migration des données.

Pour la préparation du serveur de production, nous avons réinstallé le SGBD, le serveur web Apache et Node.js, en utilisant les mêmes versions afin d'éviter des changements trop importants dans l'environnement. Cela garantit que toutes les dépendances liées à ces services soient présentes. Concernant la partie DNS, nous avons maintenu le serveur de développement comme serveur DNS. Seule modification apportée : dans le Network Manager du serveur de production, nous avons mis à jour le DNS pour qu'il pointe vers le serveur de développement. Une entrée DNS et un alias ont été ajoutés pour le serveur de production.

En ce qui concerne la migration des données, la base de données "les_brodeurs" a été arrêtée, suivie d'une sauvegarde de celle-ci ainsi que de ses utilisateurs. Le fichier de sauvegarde a ensuite été transféré vers le serveur de production, et la base de données a été restaurée sur le SGBD PostgreSQL du serveur de production. Pour le serveur Apache, seuls les sites actifs ont été migrés et réactivés. L'API Python pour le traitement des images a été redéployée sur le serveur de production, au même emplacement que sur le serveur de développement, à l'aide de son script dédié. Les fichiers de configuration et les scripts des services systèmes ont été transférés aux mêmes emplacements que sur le serveur de développement, et les services ont été redémarrés. Enfin, pour la migration du site, un dernier "pull" a été effectué sur le serveur de production, et deux fichiers de configuration ont été modifiés : le fichier **import.js** (gérant la connexion entre le site web et l'API) et le fichier **config.json** (gérant la connexion entre le site web et la base de données).

En conclusion, le serveur de production est désormais similaire pour l'essentiel, au serveur de développement à l'exception des utilisateurs qui n'ont pas été créer (utilisation d'un compte générique).

4.4.7 Test sur l'intégration des conteneurs dans l'architecture actuel

L'objectif d'utiliser des conteneurs et plus précisément Docker est de résoudre les problèmes de cohérence entre les environnements de développement et de production.

Grâce à la conteneurisation, il serait possible d'isoler les différents environnements sur le serveur. Par exemple, une application pourrait utiliser Python 3.6 dans un conteneur et Python 3.9 dans un autre, sans qu'il y ait de conflits liés aux versions des langages ou des bibliothèques.

La conteneurisation offrirait également une meilleure portabilité. Une fois qu'un composant est conteneurisé, il peut être redéployé indéfiniment et sur n'importe quel environnement compatible, sans nécessiter de reconfiguration complexe à chaque déploiement.

En termes de scalabilité, cette approche s'avère particulièrement avantageuse. Étant donné que notre application est conçue en micro-services, un service très sollicité peut être dupliqué dans plusieurs conteneurs identiques pour équilibrer la charge. Par exemple, cela pourrait être utile pour le traitement d'images si ce service est fortement sollicité.

Enfin, la conteneurisation simplifie la gestion pour les administrateurs systèmes. Lorsqu'un processus est exécuté dans un conteneur, si ce processus s'arrête, le conteneur lui-même est également arrêté. Avec un orchestrateur de conteneurs comme Kubernetes, le conteneur sera automatiquement redémarré, éliminant ainsi le besoin de surveiller manuellement la durée de vie des processus et améliorant la résilience des services.

À ce stade, l'équipe DevOps a réussi à conteneuriser le serveur web Apache. Pour ce faire, un **Dockerfile** personnalisé a été créé. L'objectif était d'inclure dans le conteneur tous les fichiers de configuration nécessaires, sans dépendre de volumes externes. Cependant, une étape cruciale reste à accomplir pour assurer le bon fonctionnement du conteneur : établir une communication efficace avec les éléments extérieurs au conteneur. Il faudrait par la suite conteneuriser le reste des éléments cité ci-dessus et créer un orchestrateur de conteneur.

Cette approche permettrait une meilleure résilience des services, ainsi qu'une scalabilité facilitée en cas de forte charge.

5 Conclusion et perspectives

5.1 Adéquation des résultats obtenus

Les résultats obtenus au cours de ce semestre sont conformes aux attentes initiales dans la plupart des domaines. Le projet a atteint une étape fonctionnelle avec la mise en place des principales composantes :

- Le traitement d'image et la conversion en patrons, qui fonctionnent grâce à l'algorithme K-means.
- Les bases de l'infrastructure technique, notamment la conteneurisation avec Docker et la gestion des requêtes via une API backend en Node.js.
- La conception des premières interfaces web pour naviguer sur le site et interagir avec les fonctionnalités.

Cependant, des ajustements sont nécessaires pour perfectionner les performances et enrichir l'expérience utilisateur.

5.2 Améliorations et objectifs pour les semestres suivants

Afin d'approfondir les fonctionnalités existantes et de répondre aux attentes des utilisateurs, plusieurs améliorations et objectifs ont été identifiés pour les semestres suivants. Ces évolutions concernent tant les aspects techniques que fonctionnels de la plateforme.

- **Automatisation des tags** : Actuellement, les recherches reposent sur des tags saisis manuellement. Une idée serait de développer et entraîner un modèle capable d'analyser les images pour générer automatiquement des tags pertinents. Cela pourrait également améliorer le système de recommandation.
- **Ouverture de l'API** : Envisager la création d'une API publique qui permettrait à des développeurs tiers d'intégrer nos outils de conversion d'images ou de gestion de patrons dans leurs propres projets.
- **Fonctionnalités communautaires** : Introduire des relations entre les utilisateurs, avec des systèmes d'amis, de discussions privées et de collaboration sur des projets communs. Ces fonctionnalités visent à renforcer l'aspect social et participatif de la plateforme.
- **Amélioration du système de recommandation** : Développer un algorithme plus avancé en s'appuyant sur les interactions des utilisateurs (tags, votes, téléchargements) afin de proposer des suggestions personnalisées de patrons.
- **Amélioration des pages web** : Finaliser l'ergonomie des pages principales (Accueil, Profil, Explorer, Convertir) et introduire un tableau de bord pour gérer les patrons et les interactions entre utilisateurs.
- **Développement de paramètres de conversion avancés** : Ajouter d'autres paramètres de conversion personnalisés.

5.3 Mot de la fin

Ce projet de génération automatique de patrons de broderie au point de croix a soulevé une problématique intéressante, mobilisant des compétences variées en traitement d'image, algorithmie, développement web et gestion des données. Cette transversalité a été au cœur du projet, montrant que des domaines en apparence distincts peuvent être efficacement intégrés pour produire une solution cohérente et innovante.

D'un point de vue technique, l'algorithme **K-means** pour la réduction des couleurs, associé à des traitements d'image comme la dilatation et l'érosion, a permis une transformation fidèle des images tout en respectant les contraintes propres au point de croix. Ces traitements ont été accompagnés de validations, telles que le calcul de l'erreur quadratique moyenne (**MSE**), pour évaluer et améliorer la qualité des résultats.

La base de données, conçue de manière structurée et évolutive, a permis le stockage des informations, tandis que l'utilisation de **Node.js** pour le backend a facilité l'utilisation d'API permettant l'utilisation du travail des autres membres du groupe.

La conception d'une interface intuitive pour l'utilisateur, combinée à une gestion fluide des échanges entre les différents composants, a mis en évidence l'importance de faire collaborer des domaines divers pour créer une expérience utilisateur optimale.

Durant ce projet, Chaque membre a contribué à différentes phases, de la recherche à la mise en œuvre, en passant par le débogage et la validation. Cette organisation a permis de coordonner des compétences variées, qu'il s'agisse de traitement d'image, de programmation web ou de gestion des infrastructures logicielles.

À l'avenir, ce projet pourrait être enrichi par l'ajout de fonctionnalités supplémentaires, comme l'estimation précise des longueurs de fils nécessaires, des outils de personnalisation avancés pour les utilisateurs, et des optimisations pour gérer efficacement des bases de données plus volumineuses. Ces évolutions viendraient renforcer encore davantage l'impact et la portée de cette application. À l'avenir, nous avons pour projet de conteneurisé notre application avec **Docker** afin de permettre une modularité et une cohérence entre les environnements de développement, et de production.

Ce projet représente une excellente démonstration de l'importance de la collaboration interdisciplinaire en informatique. L'ensemble du groupe est fier du résultat obtenu.

A Annexes

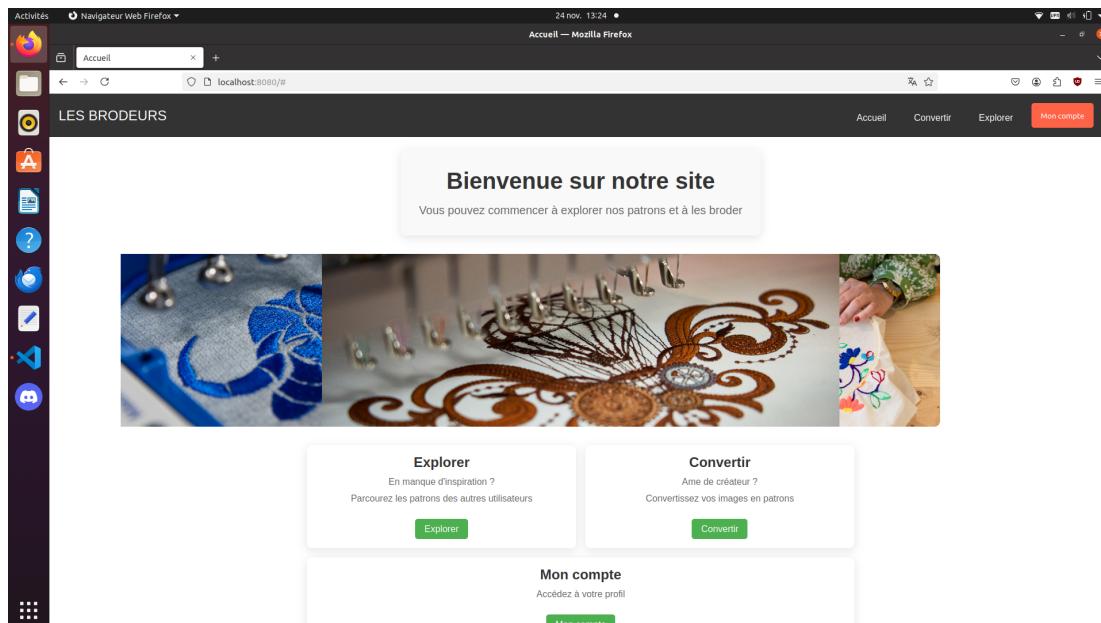


FIGURE A.1 – Page d'accueil

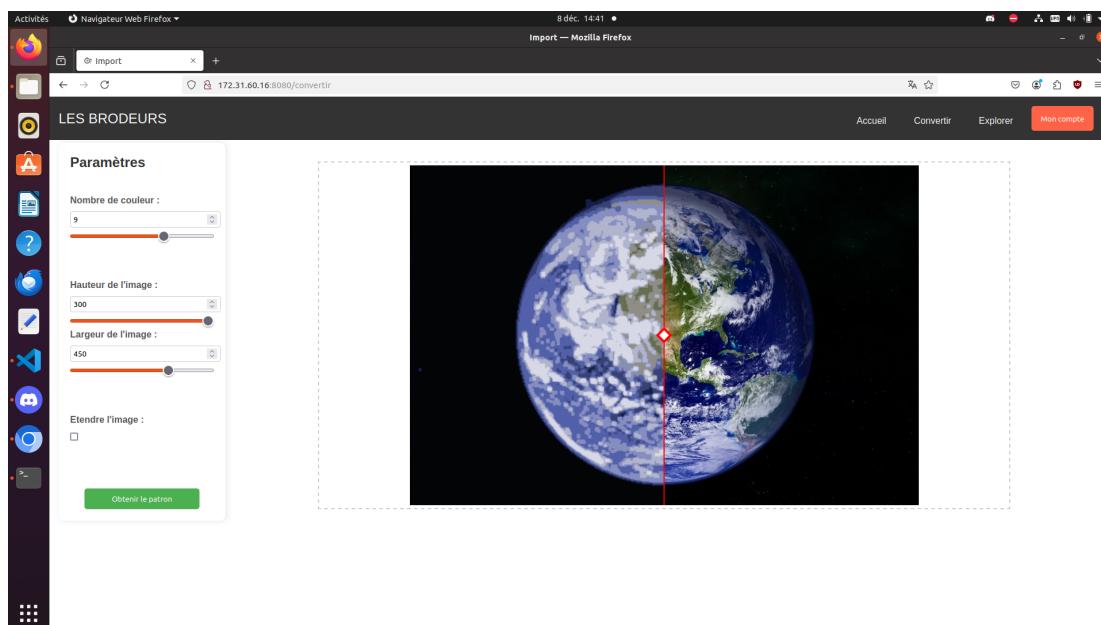


FIGURE A.2 – Page de conversion d'image en patron

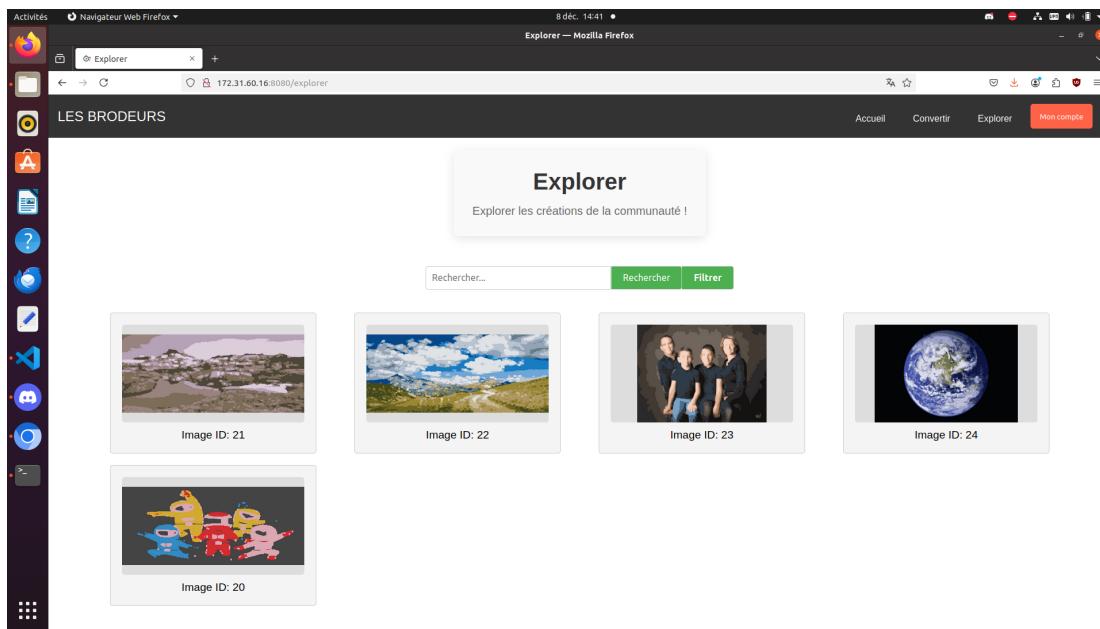


FIGURE A.3 – Page d’exploration des patrons

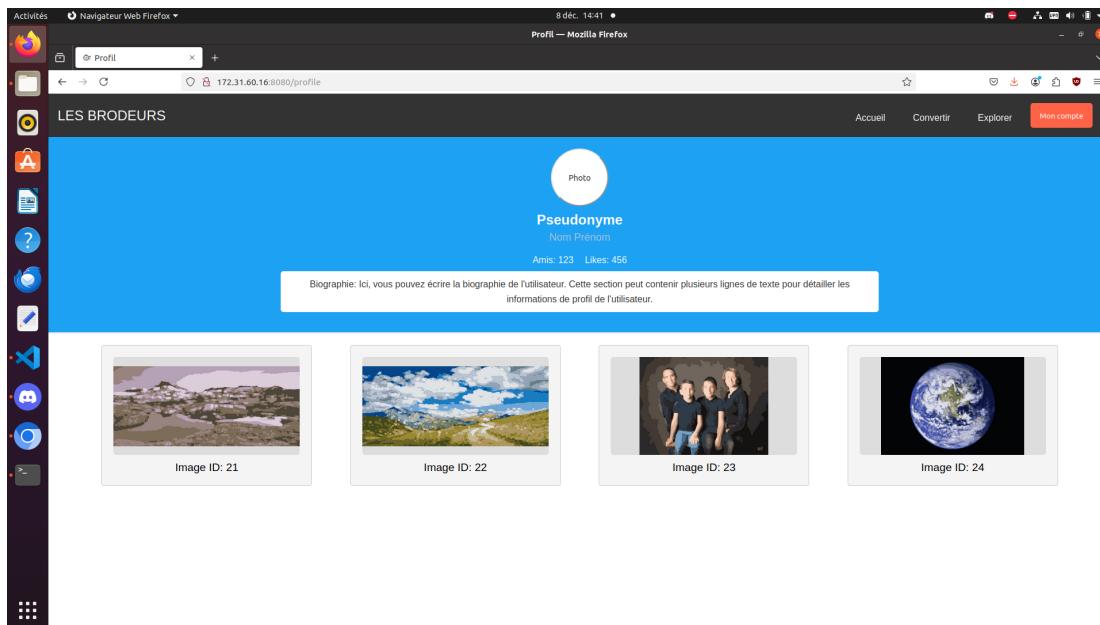


FIGURE A.4 – Page de profil