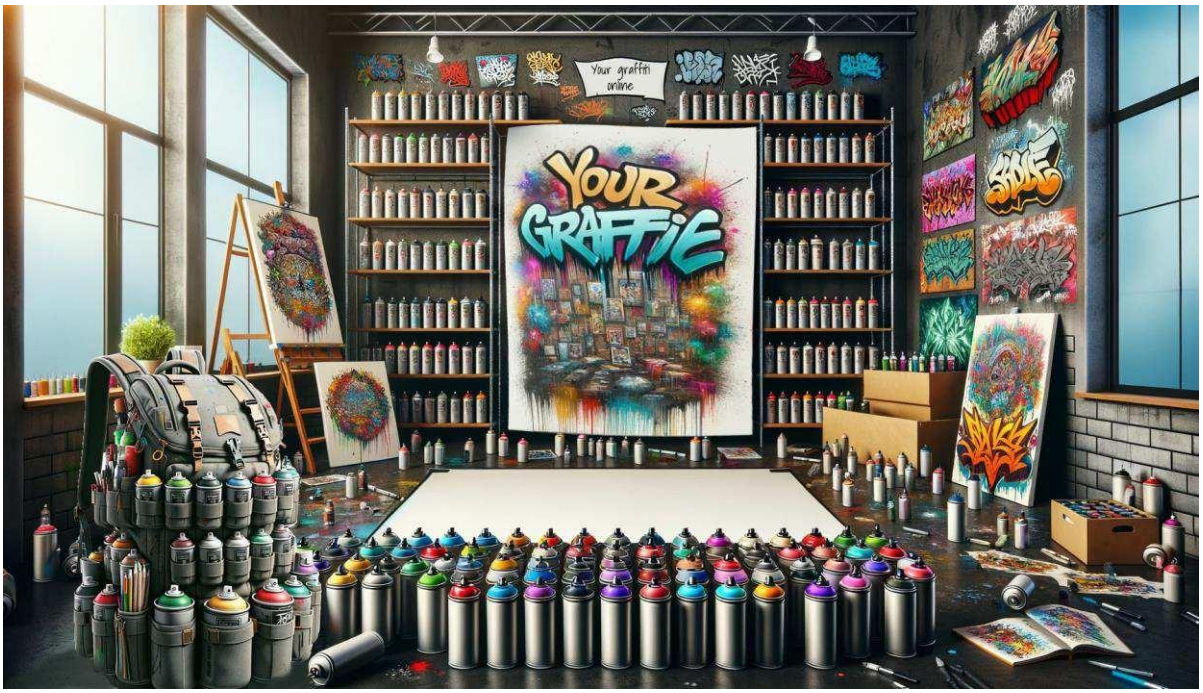
cpnv
Centre professionnel du Nord vaudois
Filière d'informatique

Projet WEB
« Your graffie »



Maxime Borgeaud
Valentin Maurer
Yasin Akyüz

Table des matières

1 Introduction

1.1 Cadre, description et motivation

Nous avons décidé de créer une petite application de création de dessin ou les artistes peuvent générer des images, les enregistrer sur la plateforme et peuvent à tout moment télécharger ses images sur son ordinateur personnel en fichier JPG.

1.2 Organisation

	Maxime	*Valentin*	*Yasin*
Partie administration	*X*	*X*	*X*
Partie DB	*X*		
Partie Server	*X*		
Partie Canva		*X*	
Partie Index		*X*	*X*
Partie Gallery	*X*		*X*
Mise en page générale			*X*

1.3 Objectifs

- ⇨ Génération des images grâce à des balises Canva et des outils personnalisés
- ⇨ Gestion des images cotés server
- ⇨ Gestion des utilisateurs sur le server
- ⇨ Gestion des droits d'accès des images selon identification des utilisateurs

2 Analyse

Le but du site est de pouvoir créer des images (style Paint) grâce à des outils personnalisés. Nous pouvons choisir des couleurs personnalisées, la taille du pinceau est également réglable. Il y a à disposition une gomme, un « undo » et un « restart ». Une fois l'image créé, si l'utilisateurs est connecté il peut l'enregistrer dans l'application (sur le server). En tant que utilisateurs nous n'avons accès qu'à nos propres images et les images « public ».
Lors de la création d'images nous pouvons lui associer des catégories.

Il faut avoir un compte, pour le créer nous devons, dans le formulaire d'inscription, renseigner : nom, prénom, email et mot de passe (le mot de passe est enregistré en hash). Une adresse email ne peut être associée à un seul compte. Afin de se connecter à son compte il faut renseigner l'émail et le mot de passe dans le formulaire de login. Une fois connecté l'utilisateurs peut modifier ses informations de comptes ou même supprimer son compte.

Notre site web se compose de 3 pages dont chacune est associée à un fichier JS et CSS :

- ⇨ Index.html
- ⇨ Canva.html
- ⇨ Gallery.html

2.1 Index.html

Dans cette page, si l'utilisateurs est connecté ou non, l'utilisateur n'a pas accès aux mêmes actions :

- ⇨ Non-connecté :
  - Login
  - Subscribe
  - Public gallery
- ⇨ Connecté :
  - Public gallery
  - My gallery
  - Logout
  - Modify account
  - Delete account

Pour sélectionner les couleurs que l'artiste va utiliser, il doit cliquer sur la partie du background qui contient la peinture en spray, cela ouvre un formulaire de sélection de couleurs. Ensuite, pour aller sur la page de canva, il faut cliquer sur le sac à dos du background. Cela ouvre un formulaire de confirmation, il faut cliquer sur « Confirmer et continuer » et l'utilisateur sera redirigé vers la page canvas.html. Que l'utilisateur soit connecté ou non, il peut accéder à la page canvas.html

2.2 Canvas.html

Lors de l'ouverture de cette page, on voit dans la section de gauche les couleurs sélectionnées précédemment. Ensuite on peut librement créer notre image avec les outils à disposition.
Il y a un radioButton pour définir le statut de l'image (par défaut sur « private »). Il y a également des checkbox pour associer des catégories à l'image (optionnel)
Ensuite si l'utilisateur est connecté, il peut cliquer sur « save » ce qui enregistrera l'image sur le server, le statut et les catégories seront liées à l'image. Si l'utilisateurs n'est pas connecté, au click de « save », la page canva se fermera et rien n'est enregistré sur le serveur.

2.3 Gallery.html

Sur la page « index.html », on peut cliquer sur « public gallery » ou « my gallery »(seulement si l'utilisateur est connecté), qui redirigent l'utilisateur sur gallery.html. Selon le bouton sur lequel l'utilisateur a cliqué, il envoie une requête qui va soit :

1) Chercher toutes les images du serveur qui ont un statut « public ».
2) Chercher toutes les images du server qui sont associées à ce user.

Dans les 2 cas les images sont affichées une par une (de la plus ancienne à la plus récente) avec les catégories correspondantes. On peut naviguer avec des bouton « next » et « back ». Si aucune image n'est trouvée un popup s'affiche pour prévenir l'utilisateur. Dans le cas de « my gallery » un bouton « delete picture » est affiché (et non sur « public gallery »).

2.4 Stratégie de test

2.4.1 Tests backends

- ⇨ Tests d'enregistrement d'un nouvel utilisateur
  - Toutes les données s'insèrent dans la db correctement ?
  - Le mot de passe est-il bien enregistré crypté ?
- ⇨ Tests de login d'un User
  - Si on entre le bon username(email) et password, se connecte il correctement
  - Si on rentre le bon username mais le mauvais password, y a-t-il le comportement attendu ?
  - Si on rentre un mauvais username, y a-t-il le comportement attendu ?
- ⇨ Tests d'enregistrement des images
  - L'image enregistrée sur le server est bien un fichier .jpg lisible par l'ordinateur ?
- ⇨ Tests de suppression d'image
  - L'image est-elle correctement supprimée du server ?
  - Tous les champs correspondants dans les tables « picture » et « picture_has_category » sonr ils bien supprimés ?

2.4.2 Tests Frontend

- ⇨ Tests d'affichage selon authentification de l'utilisateurs *(index.html)*
  - Si aucuns utilisateurs n'est connecté, les bons boutons s'affichent t ils ?
  - Si l'utilisateur est identifié, les bons boutons s'affichent t ils ?
- ⇨ Tests d'affichage des images *(gallery.html)*
  - Est-ce qu'on reçoit les bonnes données lors de la réponse du server ? (images de l'utilisateurs, images qui a statut « public »
  - Est-ce que les images s'affichent correctement à l'écran ?
  - Est-ce que la navigation entre les images de la gallerie s'éffectue correctement et sans bugs ?

3 Implémentation

3.1 Démarrer le projet

Prérequis : Il faut avoir NodeJS installé sur l'ordinateur pour pouvoir faire tourner le programme.

- ⇨ La première chose à faire est d'importer le projet depuis github avec la commande :

  Git clone https://github.com/max44444444444/projet_web.git

- ⇨ Ensuite il faut vous placer dans le dossier « backend » du projet et faire la commande « *npm start* », voici ce que vous devriez voir affiché :

```
PS C:\REPOSITORIES\projet_web\projet_web\backend> npm start

> backend@1.0.0 start
> nodemon app.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
server demmarré sur le port 3000
connexion to db successfull
```

3.2 Choix techniques

Nous avons choisi faire toute la gestion des données et d'affichage en Javascript. Le dialogue entre le server et le client se fait au moyen de requêtes http, des données lui sont parfois attachées dans le corps (utilisation de la méthode fetch). Nous avons utilisé le framework NodeJS.

Voici la liste des dépendances backend installées pour le backend :

```
"dependencies": {
  "bcrypt": "^5.1.1",
  "body-parser": "^1.20.2",
  "cors": "^2.8.5",
  "express": "^4.18.2",
  "mariadb": "^3.2.2",
  "nodemon": "^3.0.1",
  "sequelize": "^6.35.0"
}
```

- ⇨ **Bcrypt :** Il s'agit d'une bibliothèque permettant de crypter des chaines de caractères (hashage) et peut comparer une chaine cryptée avec une non-cryptée afin de vérifier s'il y a une correspondance. (login d'utilisateurs)
- ⇨ **body-parser :** c'est un middleware qui traduit automatiquement les données qui transitent en JSON.
- ⇨ **Cors :** il s'agit d'un module de NodeJS qui vérifie les autorisations des requetes http que le server reçoit (par exemple refuser les requetes POST mais accepter les GET)
- ⇨ **Express :** c'est la librairie qui permet de créer un server WEB, paramétrer le port d'écoute, définir les endpoints etc
- ⇨ **Mariadb :** Afin de pouvoir faire les requetes SQL correctement il faut installer ce module.
- ⇨ **Nodemon :** Ce module redémarre automatiquement le server à chaque CTRL+S (sinon il faut le faire manuellement à chaque fois).
- ⇨ **Sequelize :** Il s'agit d'un ORM (connexion avec la base de données. C'est ce module qui contient les méthodes qui génèrent des requêtes SQL (Dossiers « db » et « models » et également les query dans les endpoints)
- ⇨ **Canvas:** Le "canvas" est une toile numérique sur laquelle on peut dessiner avec la souris. On choisit des outils pour tracer des lignes, changer les couleurs et effacer des parties.Tout est enregistré au fur et à mesure, et on peut même revenir en arrière si on fait une erreur. Quand le résultat nous satisfait, nous pouvons enregistrer. Tout est fait en Javascript.

```
function adjustMapCoords() {
    var image = document.querySelector('#interactive-image img');
    var areas = document.querySelectorAll('#interactive-image area');

    areas.forEach(function(area) {
        var originalCoords = area.getAttribute('data-original-coords').split(',');
        var adjustedCoords = originalCoords.map(function(coord, index) {
            return index % 2 === 0
                ? coord * image.clientWidth / image.naturalWidth
                : coord * image.clientHeight / image.naturalHeight;
        });
        area.coords = adjustedCoords.join(',');
    });
}

window.addEventListener('load', adjustMapCoords);
window.addEventListener('resize', adjustMapCoords);
```

Cette fonction `adjustMapCoords` est utilisée pour ajuster les coordonnées d'une carte d'image interactive sur une page web. Le fonctionnement de la fonction peut être expliqué en plusieurs étapes simplifiées :

1. Sélection de l'image :

- La ligne `var image = document.querySelector('#interactive-image img');` sélectionne l'élément `img` (l'image) situé dans la div ayant l'identifiant `interactive-image` en utilisant le sélecteur CSS `'#interactive-image img'`.

2. Sélection des zones cliquables*
- La ligne `var areas = document.querySelectorAll('#interactive-image area');` sélectionne tous les éléments `area` (zones cliquables) situés dans la même div.

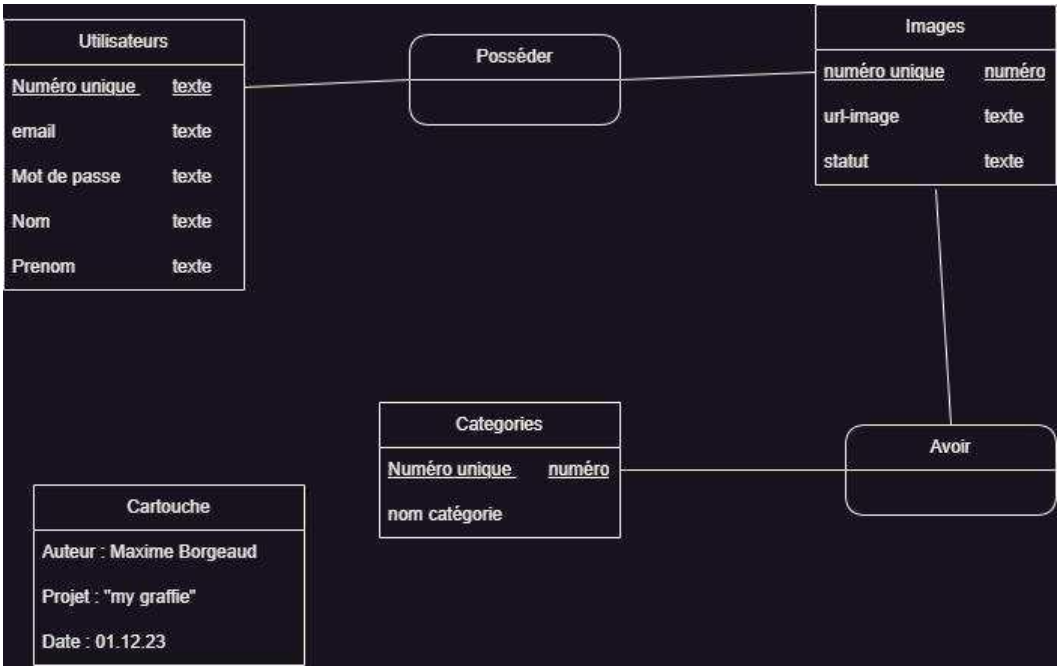
3. Ajustement des coordonnées pour chaque zone:
- La boucle `areas.forEach(function(area) {...});` est utilisée pour traiter chaque élément `area` individuellement.
- `var originalCoords = area.getAttribute('data-original-coords').split(',');` récupère les coordonnées d'origine de chaque `area` à partir de l'attribut `data-original-coords` et les convertit en un tableau séparé par des virgules.

4. Recalcul des coordonnées:
- `originalCoords.map(function(coord, index) {...})` utilise une fonction `map` pour transformer les coordonnées d'origine en fonction des dimensions actuelles de l'image.
- `return index % 2 === 0 ? coord * image.clientWidth / image.naturalWidth : coord * image.clientHeight / image.naturalHeight;` ajuste chaque coordonnée (x ou y) en fonction du rapport entre la largeur actuelle de l'image (`clientWidth`) ou la hauteur (`clientHeight`) et la largeur d'origine (`naturalWidth`) ou la hauteur d'origine (`naturalHeight`). Si la coordonnée est `x` (index pair), elle est ajustée en fonction du rapport de largeur ; si elle est `y` (index impair), elle est ajustée en fonction du rapport de hauteur. Cela garantit que les coordonnées restent correctes lorsque l'image est mise à l'échelle en fonction de la taille de l'écran ou de la conception réactive.

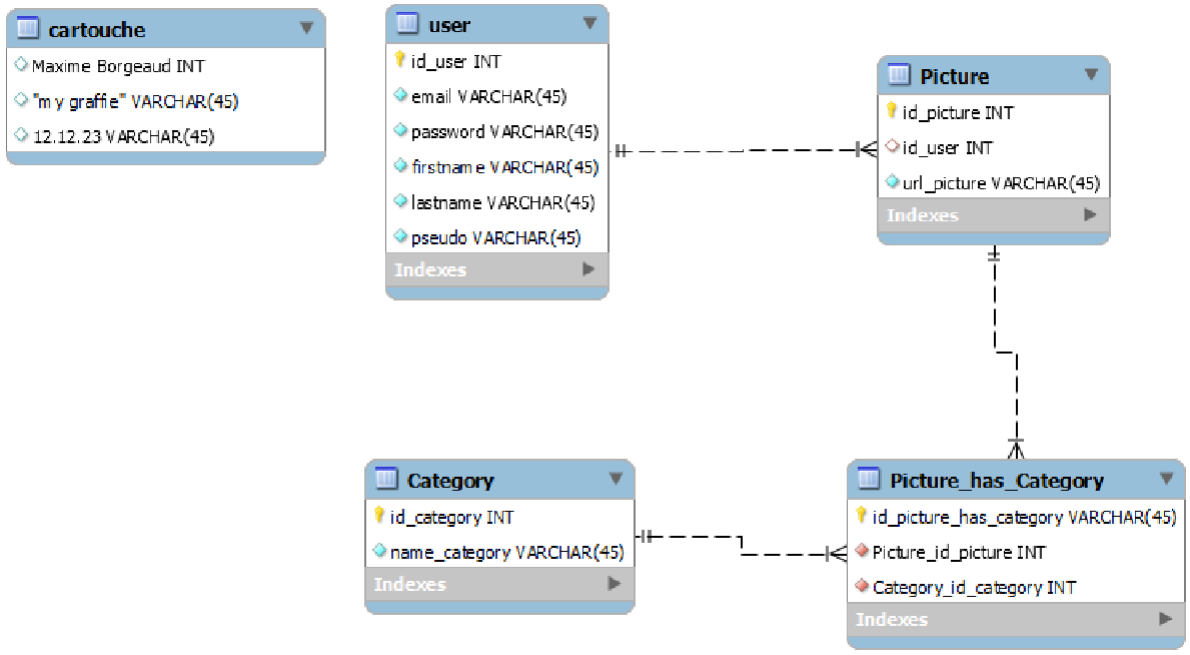
5. Application des nouvelles coordonnées :
- `area.coords = adjustedCoords.join(',');` assigne les nouvelles coordonnées calculées à l'attribut `coords` de chaque élément `area`. La fonction `join(',')` est utilisée pour fusionner les coordonnées en une chaîne de caractères, de manière à ce qu'elles soient correctement formatées pour chaque élément `area`.

Cette fonction permet de garantir que la carte d'image interactive fonctionne correctement lorsque la taille de la page change, par exemple lors du redimensionnement de la fenêtre ou lors de l'affichage sur un appareil mobile. L'ajustement dynamique des coordonnées permet aux zones cliquables de toujours correspondre aux bonnes positions de l'image, ce qui est particulièrement important dans le cadre de la conception web réactive, où les images peuvent être redimensionnées de différentes manières en fonction de la taille et de la résolution de l'écran.

3.3 Modèle conceptuel de données



3.4 Modèle Logique de données



3.5 Points techniques spécifiques

3.5.1 Enregistrement et affichage des images

Toute communication entre le serveur et le client se fait via des requêtes http faites avec la fonction « fetch() » Javascript. Le transite des images fut le plus compliqué car il faut transformer les fichiers .jpg en chaines de caractères et les mettre dans le corps de la requête. Ces données étant si grandes il fallut modifier la taille limite par défaut autorisées.

3.6 Bugs

Lorsque l'on est dans la gallerie d'image de l'utilisateur, si l'on supprime cette seule image, elle ne disparait pas de l'écran alors que elle est bien supprimée du server et de la db.

Il se trouve que lors de mon tout dernier essai avant de vous rendre le projet je n'ai pas réussi à démarrer le serveur Express. (mais peut-être est-ce du au configurations de mon ordinateur). Il est donc possible que si vous suivez la procédure pour démarrer le projet vous vous trouviez devant cette erreur :

```
[nodemon] starting `node app.js`
node:internal/modules/cjs/loader:1340
  return process.dlopen(module, path.toNamespacedPath(filename));
                 ^

Error: \\?\E:\REPOSITORIES\projet_web\backend\node_modules\bcrypt\lib\binding\napi-v3\bcrypt_lib.node nest pas une application Win32 valide.
\\?\E:\REPOSITORIES\projet_web\backend\node_modules\bcrypt\lib\binding\napi-v3\bcrypt_lib.node
    at Module._extensions..node (node:internal/modules/cjs/loader:1340:18)
    at Module.load (node:internal/modules/cjs/loader:1119:32)
    at Module._load (node:internal/modules/cjs/loader:960:12)
    at Module.require (node:internal/modules/cjs/loader:1143:19)
    at require (node:internal/modules/cjs/helpers:110:18)
    at Object.<anonymous> (E:\REPOSITORIES\projet_web\backend\node_modules\bcrypt\bcrypt.js:6:16)
    at Module._compile (node:internal/modules/cjs/loader:1256:14)
    at Module._extensions..js (node:internal/modules/cjs/loader:1310:10)
    at Module.load (node:internal/modules/cjs/loader:1119:32)
    at Module._load (node:internal/modules/cjs/loader:960:12) {
  code: 'ERR_DLOPEN_FAILED'
}

Node.js v18.16.1
[nodemon] app crashed - waiting for file changes before starting...
```

Il semblerait qu'il y aie un problème avec le module « bcrypt » cepdandant le module « nodemon » fonctionne correctement. Peut-être que j'ai mal installé NodeJS sur cet ordinateur.

4 Conclusions

4.1 Objectifs atteints

Les fonctionnalités de création d'images sur un canva html ainsi que de les lier à des catégories et un statut puis l'enregistrement de toutes ces informations sur le server fonctionnent parfaitement. La galerie permet de naviguer parmi les images enregistrées dans le server, dans l'ordre des plus anciennes au plus récentes.
La gestion de l'authentification et des données des utilisateurs fonctionne également parfaitement.

4.2 Objectifs non-atteints

Au tout début du projet nous voulions implémenter la faite de pouvoir reprendre l'édition des images après l'enregistrement de celles-ci. Nous nous sommes assez rapidement rendu compte qu'avec le temps à notre disposition il serait très difficile d'y arriver.
La mise en production en ligne sur Swisscenter a partiellement fonctionné. L'application est en ligne et disponible à l'addresse : mygraffie.mycpnv.ch. Cependant aucunes requetes POST ne fonctionne, bloquée par le navigateur avec la politique de « CORS policy ». Erreur qui, selon nos recherches, peut-être corrigée système de tokenJWT entre le frontend et le backend.

4.3 Suites possibles au pour le projet

Premièrement palier aux objectifs non-atteints du projet serait une bonne évolution. Ensuite il serait intéressant d'améliorer la galerie publique, notamment faire une recherche par catégorie. On pourrait aussi faire que les utilisateurs puissent créer de nouvelles catégories. Il serait également possible de rajouter une multitude d'outils pour l'édition des images.

5 Annexes

Base de donnée :

Projet_web.sql

User cases :

usercases.xsls

Maquettes (balsamiq) :

Maquettesv1_myGraffie.bmpr
Maquettesv2_myGraffie.bmpr

Journals de bords :

MaximeBoard.pdf
ValentinBoard.pdf
YasinBoard.pdf