CAHIER DES CHARGES

Projet d'Atelier de Professionnalisation

Application de gestion d'images artistiques — Systeme Gestion Image (Gravitart)

Client	Gravitart — Magasin de fournitures artistiques
Prestataire	Individu libéral.
Applications	Front web PHP (ce dépôt) + back-end de gestion séparé Java (autre dépôt)
Base de données	MariaDB — `GestionSystemeImage`

1. Cadre du projet

1.1 Description du projet

Gravitart souhaite proposer une plateforme communautaire de publication d'images artistiques, inspirée des galeries visuelles en ligne. Le projet doit permettre la publication d'œuvres, leur consultation, le classement par tags et la modération des contenus.

1.2 Contexte de l'entreprise

L'entreprise veut attirer de nouveaux artistes et valoriser son image de marque, tout en gardant un contrôle éditorial sur les contenus publiés.

Enjeux principaux : - Publication simple et rapide d'images - Gestion des rôles utilisateurs - Modération (signalements, suppression, gestion des comptes) - Recherche par tags

1.3 Objectifs

Référence	Description
Obj. 1	Permettre l'inscription, la connexion et la gestion de session sécurisée
Obj. 2	Autoriser le téléversement d'images et la gestion de la visibilité (publique/privée)
Obj. 3	Permettre la recherche et la navigation des images par tags
Obj. 4	Permettre le signalement des contenus inappropriés

Référence	Description
Obj. 5	Donner aux administrateurs des fonctions de modération (signalements, comptes, suppression)
Obj. 6	S'appuyer sur une base MariaDB structurée et cohérente avec le modèle métier

1.4 Livrables

Livrable	Contenu
L1	Application web PHP (architecture MVC simple)
L2	Schéma relationnel MariaDB `GestionSystemeImage`
L3	Script SQL d'initialisation/export de la base
L4	Documentation technique consolidée (ce document)
L5	Référentiel des rôles, règles d'accès et mesures de sécurité

---

2. Conception graphique

2.1 Principes visuels

- Interface orientée galerie d'images artistiques
- Palette colorée et identité visuelle Gravitart (assets/logo et fond)
- Navigation par pages dédiées (dashboard, galerie, image, upload, suppression, comptes, signalements)

2.2 Pages principales

- `login.php` / `register.php` : authentification et création de compte
- `dashboard.php` : point d'entrée après connexion selon rôle
- `site.php` : galerie et recherche par tag
- `image.php` : détail image, tags, signalement
- `upload.php` : téléversement image (admin/artiste)
- `removeimage.php` : suppression image (admin ou propriétaire artiste)
- `listecompte.php` : liste des comptes (admin)
- `signalements.php` : consultation des signalements (admin)

---

3. Spécifications fonctionnelles

3.1 Rôles applicatifs

Rôle	Description
Admin	Modération globale : comptes, signalements, suppression d'images, édition tags
Artiste	Publication, consultation, signalement, suppression de ses propres images
Invité	Compte avec droits restreints (consultation connectée sans actions de modération/publication)

Remarque : dans l'état actuel du projet, la galerie est accessible via session connectée (pas d'accès anonyme public).

3.2 Fonctionnalités par rôle

OBJECTIF	ACTEUR	IMPACT	FONCTIONNALITÉS
Gérer l'accès à la plateforme	Tous	Compte	Inscription, connexion, déconnexion, expiration de session
Publier des œuvres	Artiste / Admin	Images	Téléversement BLOB, choix visibilité publique/privée
Explorer la galerie	Tous comptes connectés	Consultation	Liste des images, recherche par tag, affichage détail
Signaler un abus	Tous comptes connectés	Modération	Création de signalement avec motif normalisé + motif libre
Modérer le contenu	Admin	Gouvernance	Consultation signalements, liste des

OBJECTIF	ACTEUR	IMPACT	FONCTIONNALITÉS
			comptes, suppression élargie
Gérer les tags	Admin (+ artiste selon contexte actuel)	Qualité de classification	Ajout/remplacement des tags d'une image

4. Spécifications techniques

4.a Présentation de la solution

La solution est composée de : - Une application web PHP (ce dépôt) organisée en couches `controleur/`, `modele/`, `vue/` - Une base MariaDB `GestionSystemeImage` - Un back-end de gestion complémentaire (autre dépôt) orienté administration étendue (gestion utilisateurs, signalements, création de tags)

Le front web actuel couvre déjà des fonctions administratives locales : consultation des signalements, listing des comptes, actions de modération sur les images.

4.b Choix technologiques

Technologie	Usage	Justification
PHP 8+	Logique web	Simplicité de déploiement et compatibilité hébergement classique
PDO + requêtes préparées	Accès BDD	Réduction des risques d'injection SQL
MariaDB 10.x	Persistance relationnelle	Modèle structuré, contraintes FK, cohérence des données
Architecture MVC simple	Organisation code	Séparation contrôleurs/modèles /vues
Sessions PHP	Authentification	Gestion d'état utilisateur, timeout d'inactivité

Technologie	Usage	Justification
Hash mot de passe (`password_hash`)	Sécurité des comptes	Stockage non clair des mots de passe côté application

4.c Architecture applicative

i. Contrôleurs principaux

- `AuthControleur` : connexion, inscription, verrouillage anti-bruteforce, timeout session
- `ImageControleur` : listing, recherche par tag, upload, suppression, CSRF
- `UtilisateurControleur` : listing utilisateurs, gestion signalements

ii. Couche modèle / DAO

- Entités : `Utilisateur`, `Image`, `Tags`, `Signalement`, `Role`, `Signale`
- DAO : `UtilisateurDAO`, `ImageDAO`, `TagsDAO`, `SignalementDAO`, `RoleDAO`, `SignaleDAO`

iii. Flux fonctionnels clés

- Connexion : formulaire → `AuthControleur::traiterConnexion` → vérification hash mot de passe → session
- Upload : `upload.php` → `ImageControleur::traiterTeleversement` → insertion `IMAGE`
- Signalement : `image.php` → `UtilisateurControleur::traiterSignalement` → insertion `SIGNALEMENT` + `SIGNALE`
- Recherche tag : `site.php` → `ImageControleur::rechercherParTag` → jointure `IMAGE/ASSOCIE`

4.d Base de données

i. Dictionnaire de données (synthèse)

Mnémonique	Type	Sensibilité	Observation
`id_utilisateur`	INT AI	Personnelle	PK `UTILISATEUR`
`nom_utilisateur` / `prenom_utilisateur`	VARCHAR(50)	Personnelle	Identité
`mail_utilisateur`	VARCHAR(50)	Personnelle	UNIQUE
`cell_utilisateur`	VARCHAR(15)	Personnelle sensible	UNIQUE
`mdp`	VARCHAR(200)	Sensible	Mot de passe (hash attendu)
`id_role`	INT	-	FK vers `ROLE`
`id_image`	INT AI	-	PK `IMAGE`
`image`	LONGBLOB	Potentiellement sensible	Donnée binaire image

Mnémonique	Type	Sensibilité	Observation
switchpriv	TINYINT(1)	-	Visibilité image
tag	VARCHAR(100)	-	PK TAGS
id_signalement	INT AI	-	PK composite avec utilisateur
libelle_signalement	ENUM	-	Motif normalisé
libelle_signalement_custom	VARCHAR(250)	-	Complément libre

ii. Dépendances fonctionnelles (principales)

- id_utilisateur → nom_utilisateur, prenom_utilisateur, mail_utilisateur, cell_utilisateur, mdp, id_role
- id_role → libelle_role
- id_image → image, switchpriv, id_utilisateur
- tag → (valeur du tag)
- id_signalement + id_utilisateur → libelle_signalement, libelle_signalement_custom

iii. MCD (entités et associations)

Entités : UTILISATEUR, ROLE, IMAGE, TAGS, SIGNALEMENT

Associations : - ASSOCIE(id_image, tag) : liaison n,n entre images et tags - SIGNALE(id_image, id_signalement, id_utilisateur) : liaison entre image signalée, motif et auteur du signalement

iv. MLD (vue relationnelle)

- ROLE(id_role, libelle_role)
- UTILISATEUR(id_utilisateur, nom_utilisateur, prenom_utilisateur, mail_utilisateur, cell_utilisateur, mdp, id_role#)
- IMAGE(id_image, image, switchpriv, id_utilisateur#)
- TAGS(tag)
- SIGNALEMENT(id_signalement, id_utilisateur#, libelle_signalement, libelle_signalement_custom)
- ASSOCIE(id_image#, tag#)
- SIGNALE(id_image#, id_signalement#, id_utilisateur#)

v. Script SQL et ordre logique de création

1. Tables de référence : ROLE, TAGS
2. Utilisateurs : UTILISATEUR
3. Médias : IMAGE

4. Signalements : `SIGNALEMENT`
5. Tables de liaison : `ASSOCIE`, `SIGNALE`

4.e Cybersécurité

Mesure	Portée	Détail
Requêtes préparées PDO	DAO	Protection contre injections SQL
Hash mot de passe	Authentification	Vérification via `password_verify`
Contrôle d'accès par rôle	Contrôleurs	Vérification admin/artiste/invité selon action
Jeton CSRF	Formulaires sensibles	Vérification sur suppression/signalement/modification tags
Session sécurisée	Connexion	Timeout d'inactivité + régénération d'identifiant de session
Verrouillage login	Authentification	Blocage temporaire après échecs répétés

4.f Écarts et cohérence avec l'existant

- Cette documentation est alignée sur l'état réel du code et du schéma SQL présents dans le projet.

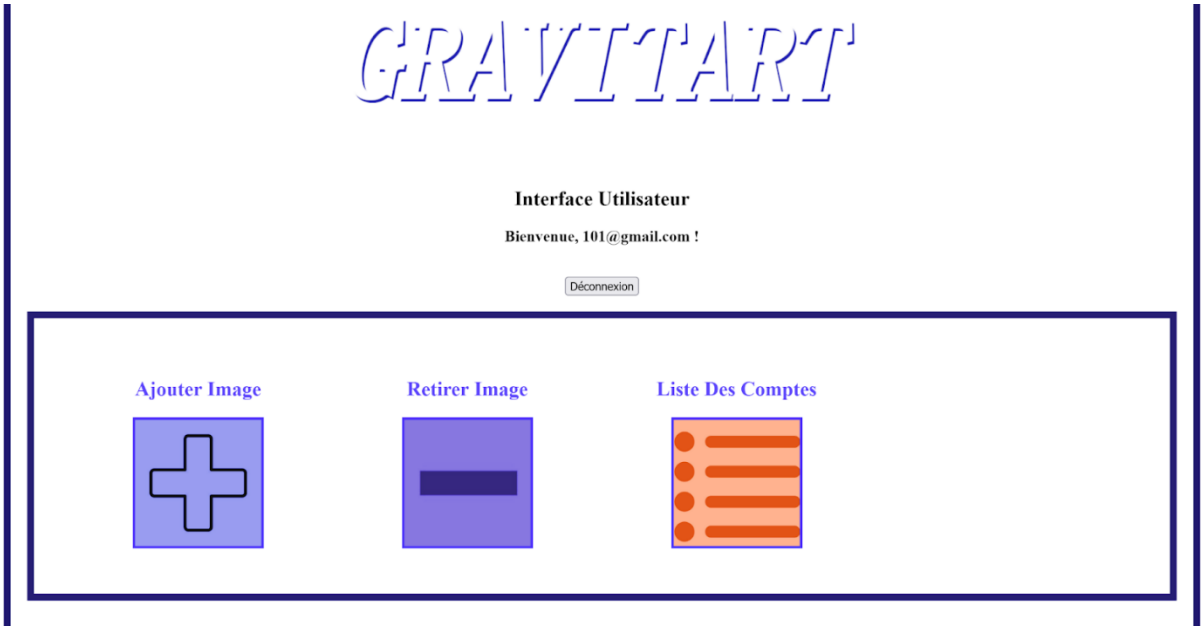
CAPTURES D'ECRAN

Vue du frontend PHP:

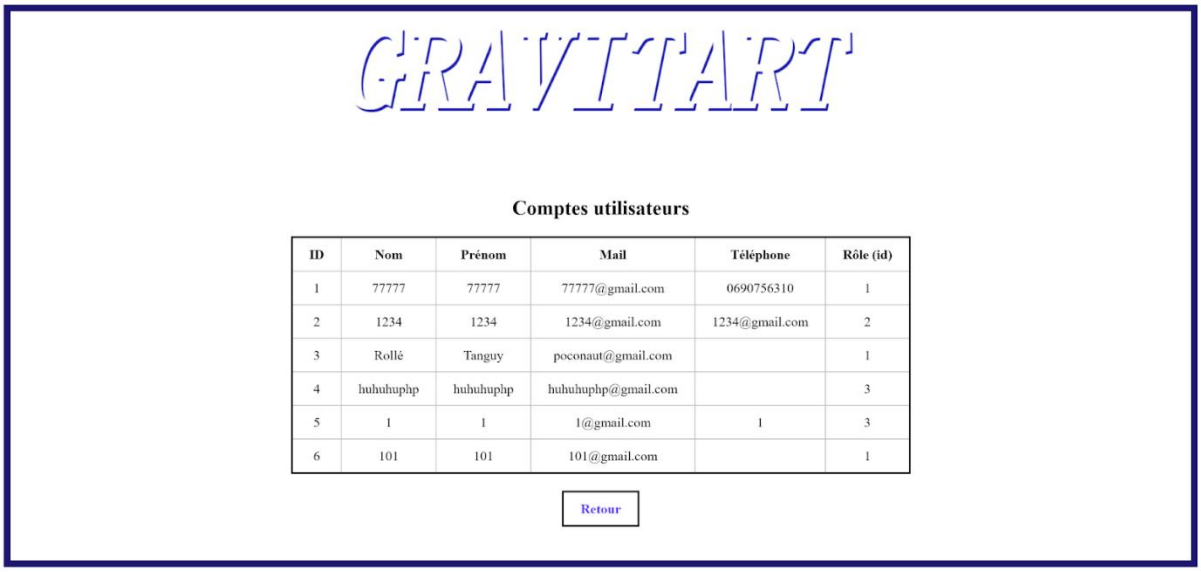
Vue du frontend, interface utilisateur php (role = ARTISTE):

A retirer de l’interface artiste et utilisateur (ou du moins les images non-propriétaires aux utilisateurs):

Interface utilisateur administrateur:



Liste Des Comptes (graphique):



MLD

UTILISATEUR(id_utilisateur, nom_utilisateur, prenom_utilisateur, mail_utilisateur, cell_utilisateur, id_role)
Clé primaire: id_utilisateur
#Clé étrangère id_role de référence ROLE

IMAGE(id_image, image, switchpriv)
Clé primaire: id_image
#Clé étrangère id_utilisateur de référence UTILISATEUR

SIGNALEMENT(id_signalement, id_utilisateur libelle_signalement, libelle_signalement_custom, id_utilisateur)
Clé primaire: id_signalement, id utilisateur
#Clé étrangère id_utilisateur de référence UTILISATEUR

TAGS(tag, id_image)
Clé primaire: tag
#Clé étrangère id_image de référence IMAGE

ROLE(id_role, libelle_role)
Clé primaire: id_role

—Relations n,n, clé étrangères exclusivement—

signale(id_image, id_signalement, id_utilisateur)
Clé primaire: id_image, id_signalement, id_utilisateur
#Clé étrangère id_image de référence IMAGE
#Clé étrangère id_signalement de référence SIGNALEMENT

CREATE TABLE ROLE (

id_role NOT NULL AUTO_INCREMENT,

libelle_role varchar(50) NOT NULL,

PRIMARY_KEY (id_role)

);

CREATE TABLE UTILISATEUR (

id_utilisateur NOT NULL AUTO_INCREMENT,

nom_utilisateur varchar(50) NOT NULL,

prenom_utilisateur varchar(50) NOT NULL,

mail_utilisateur varchar(50) NOT NULL UNIQUE,

cell_utilisateur varchar(15) UNIQUE,

id_role NOT NULL AUTO_INCREMENT,

PRIMARY KEY (id_utilisateur),
FOREIGN KEY (id_role) REFERENCES ROLE (id_role)

);

CREATE TABLE IMAGE (

id_image NOT NULL AUTO_INCREMENT,

image LONGBLOB,

switchpriv BOOLEAN,

id_utilisateur NOT NULL AUTO_INCREMENT,

PRIMARY KEY (id_image),
FOREIGN KEY (id_utilisateur) REFERENCES UTILISATEUR (id_utilisateur)

);

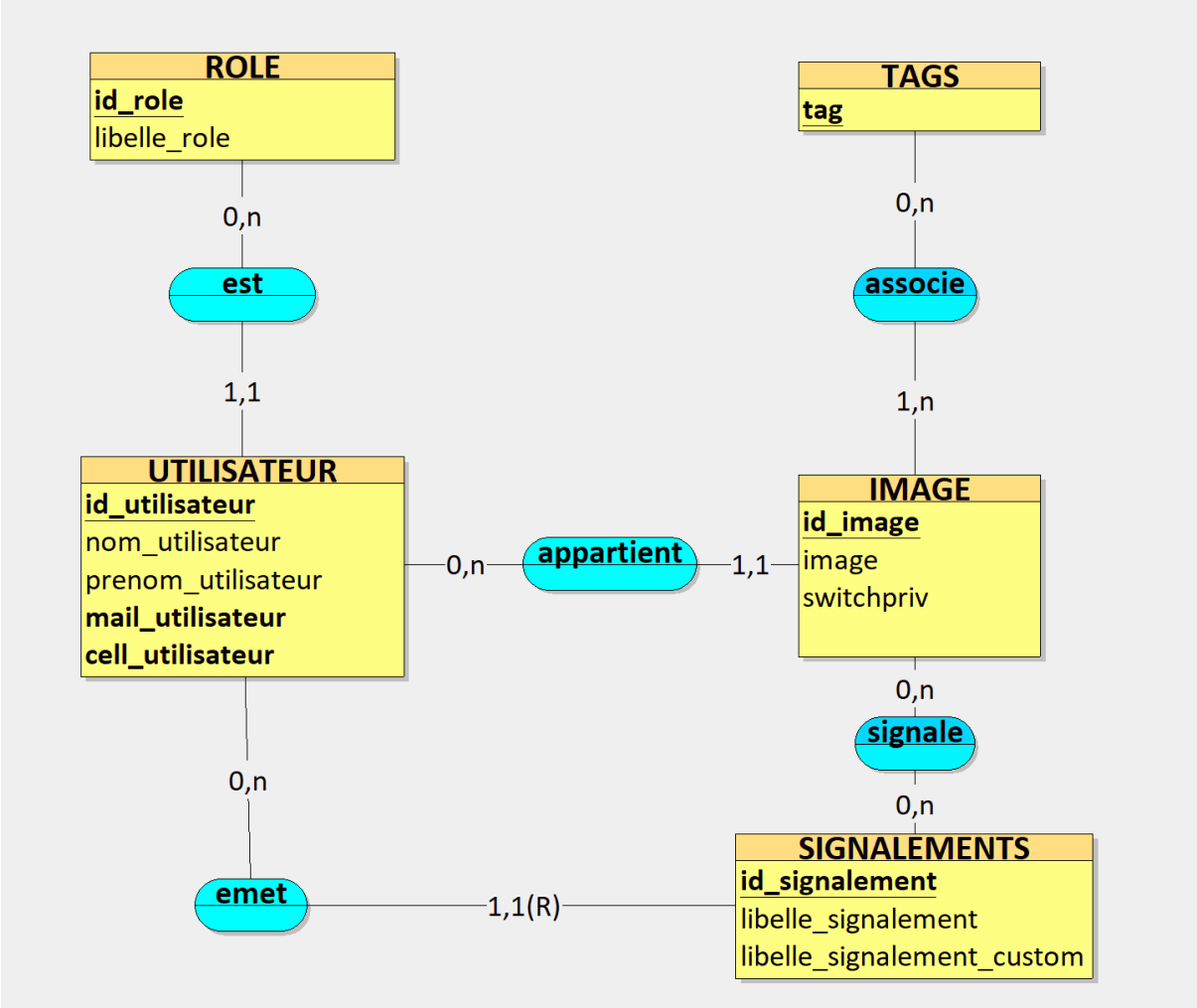
CREATE TABLE SIGNALEMENT (

id_signalement NOT NULL AUTO_INCREMENT,

id_utilisateur NOT NULL AUTO_INCREMENT,

```
libelle_signalement enum('violence', 'haine', 'contenu_sexuel', 'contenu_illegal', 'plagiat',
'autre') NOT NULL,

libelle_signalement_custom varchar(250),

PRIMARY KEY (id_signalement, id_utilisateur),
FOREIGN KEY (id_utilisateur) REFERENCES UTILISATEUR (id_utilisateur)

);

CREATE TABLE TAGS (

tag varchar(100) NOT NULL UNIQUE,

id_image NOT NULL AUTO_INCREMENT,

PRIMARY KEY (tag),
FOREIGN KEY (id_image) REFERENCES IMAGE (id_image)

);



CREATE TABLE signale(

id_image NOT NULL AUTO_INCREMENT,

id_signalement NOT NULL AUTO_INCREMENT,

id_utilisateur NOT NULL AUTO_INCREMENT,

PRIMARY KEY (id_image, id_signalement, id_utilisateur),
FOREIGN KEY (id_image) REFERENCES IMAGE (id_image),
FOREIGN KEY (id_signalement) REFERENCES IMAGE (id_signalement),
FOREIGN KEY (id_utilisateur) REFERENCES IMAGE (id_utilisateur)


);
```

A noter qu'une erreur conceptuelle fut repérée vers la fin du projet, concernant la gestion des tags d'images. La cardinalité 1,1 de tags à image fut changée en une relation 0, n et une nouvelle association fut ajoutée dans le MCD, MLD et le script: soit:



//
TAGS(tag, id_image)
Clé primaire: tag

associe(id_image, tag)
Clé primaire: id_image, tag
#Clé étrangère id_image de référence IMAGE
#Clé étrangère tags de référence TAGS

//

CREATE TABLE TAGS (

tag varchar(100) UNIQUE,

PRIMARY KEY (tag)

```
);

CREATE TABLE ASSOCIE (

id_image INT,

tag varchar(100) UNIQUE,

FOREIGN KEY (id_image) REFERENCES IMAGE(id_image),
FOREIGN KEY (tag) REFERENCES TAGS(tag),
PRIMARY KEY (id_image, tag)

);
```