

**DOSSIER DE PROJET FIL ROUGE**

## Développeur web et web mobile

**Paul VIAU**



## Remerciements

Je tiens tout d’abord à exprimer ma reconnaissance envers le formateur référent de ma promotion, Mathieu Mithridate ainsi que la coordinatrice de ma formation, Florence Calmettes pour leur accompagnement, leur disponibilité et leur patience à toutes épreuves tout au long de cette formation.

Je remercie également l’ensemble des formateurs et des intervenants de la formation qui ont assuré la partie théorique de celle-ci.

Sans oublier M. Jean-Yves Artigala qui a été mon tuteur de stage à distance pendant deux mois au sein de l’association des chanteurs montagnards de Bagnères de Bigorre.

Enfin je remercie l’ensemble de mes camarades de promotion, où l’entraide et la bienveillance auront été les mots d’ordre.

# SOMMAIRE

Abstract

Introduction

Compétences du Titre couvertes par le Projet

Analyse du Besoin

Spécifications Fonctionnelles :

- Use Case

- Diagramme d’activité

- Diagramme séquence

- Maquettage

- Arborescence

CONCEPTION :

- MDC

- MLD

Outils Techniques Utilisés

Fonctionnalités

- Ecran

- Code

CONCLUSION

ANNEXES

- SQL

# **ABSTRACT**

My name is Paul Viau and I am 24 years old. I currently live in Toulouse but I grew up in Madrid till I was 10 years old and then I moved to the Pyrenees, in Bagnères de Bigorre, where I stayed for eight more years.

I studied Linguistics for three years and after that I did a two years sound technician and audio producer training. I really enjoyed it since music is my passion but it was really hard and stressful to find jobs as an intermittent in the performing arts. That’s why I had to find another career, where employment security is more stable.

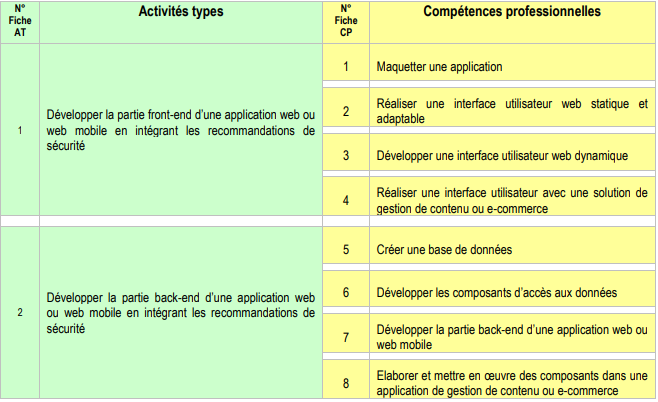
I‘ve always been interested in information technology because I produce a lot of music on my computer and development intrigued me increasingly. I was very interested in how the software I used really worked. Some of my friends are developers so I asked them what their job looked like. I spent two weeks with one of them who works as a freelance web developer to see the working routine of a developer. Another friend told me about subsidized development trainings and that’s when I started applying to training centers like ADRAR Pole Num, who accepted my application.

During the training I worked on a personal project which is called Diggin Nuggets. It’s a website where the users can post and discover music via youtube links. There is also a magazine section where you can find articles mostly about electronic music. This is the project that I will introduce in order to show my abilities.

# **INTRODUCTION**

Je m’appelle Paul Viau, j’ai 24 ans, je suis en reconversion professionnelle et j’aspire à devenir développeur. Avant la formation j’exerçais en tant que technicien du son en but d’obtenir l’intermittence du spectacle. Ayant été qualifié en 2020, J’avais beaucoup de peine à obtenir des contrats d’intermittence du spectacle en partie à cause de la crise sanitaire. L’instabilité mais aussi la pénibilité du métier m’ont fait me tourner de plus en plus vers mon deuxième gros centre d’intérêt après la musique : l’informatique. En effet, plusieurs amis développeurs m’ont conseillé de suivre des formations qualifiantes comme celles que propose l’ADRAR. C’est ainsi que j’ai passé les tests et été admis dans la structure.

Le projet que je vais vous présenter été réalisé pendant la formation et a pour but de valider les compétences suivantes :



Avec ce projet je vais essayer de vous démonter que je suis capable de maquetter une application, réaliser une interface utillisateur web statique et adaptable, créer une base de données, développer les composants d’accès aux données et développer la pertie back-end d’une application web ou web mobile.

## Presentation du projet :

Il s’agit d’un site web nommé Diggin Nuggets. C’est un site de partage et de découverte de musique où les utilisateurs disposant d’un compte peuvent poster des musiques via lien Youtube avec une petite description du morceau et spécifient le style musical du morceau. Ces données sont enregistrées dans la base de données et n’importe quel utilisateur peut trier les publications en fonction du style recherché. Les utilisateurs peuvent « aimer » et commenter les publications.

Le site dispose également d’une partie magazine avec des articles que les utilisateurs peuvent aussi visiter, liker et commenter

# **ANALYSE DU BESOIN**

La start-up toulousaine Diggin Nuggets souhaite lancer un site web où les utilisateurs peuvent collaborer afin de partager et découvrir des nouvelles musiques.

L’opportunité que présenterait ce site serait d’y proposer des publicités afin de le rentabiliser .

Le client souhaite un site où les utilisateurs peuvent publier sur un fil d’actualité des liens Youtube convertis en incrustations accompagnés d’une description et de la spécification du genre musical. Les utilisateurs doivent pouvoir mentionner qu’ils aiment une publication et la commenter. Le site proposera également une partie magasine avec des articles que les utilisateurs peuvent aimer et commenter. Pour la partie graphique il est demandé une interface futuriste aux couleurs froides mais un fond rappelant le thème du site, la musique.

Cibles :

Les cibles sont principalement des djs à la recherche de nouveautés musicales à proposer à leur public mais aussi des passionnées de musique, principalement électronique, meme si l’on ne peut pas révoir ce que vont poster les utilisateurs.

Ce projet implique de développer un système d’information muni d’une base de données relationnelle conceptualisée grâce à la méthode merise.

Il sera également être développé orienté objet afin de faire interagir les entités entre elles.

Police à utiliser : michroma.

couleur de la police : #293241

Image en arrière-plan représentant des disques vinyles

# **SPECIFICATIONS FONCTIONNELLES**

Pour comprendre, définir et représenter le fonctionnement d’une application, il est d’usage de réaliser et d’utiliser des diagrammes de type UML (Utility Modeling Language). Il s’agit d’un langage de modélisation qui permet de réaliser des diagrammes normalisés qui décriront plus ou moins en détail la structure ou le comportement d’un système d’information.

Il existe 14 types de diagrammes UML mais je ne vais en présenter que trois pour ce projet. Le diagramme de cas d’utilisation, le diagramme d’activité et le diagramme de séquence. Ce sont tous les trois des diagrammes comportementaux.

## Le diagramme de cas d’utilisation (use case) :

Le diagramme de cas d’utilisation, dit use case, liste toutes les fonctionnalités d’une application en fonction des droits qu’a l’acteur qui l’utilise. Un acteur représente n’importe quelle entité au système qui interagit avec. Chaque bulle représente une fonctionnalité de l’application. Ici, il y a quatre acteurs : le visiteur, l’utilisateur, le rédacteur et l’administrateur. Ils n’ont pas tous les mêmes droits et c’est ce qui va être défini dans ce diagramme.

## 

* **Le visiteur :** c’est un utilisateur non identifié. Il peut se connecter, s’inscrire, lire les articles, voir les publications, visiter le profil des autres utilisateurs et trier les publications en fonction du style recherché.
* **L’utilisateur**: c’est un utilisateur identifié. La flèche qui le relie au visiteur est une Generalization, elle représente un héritage des droits. Il hérite des droits du visiteur et en a d’autres : se déconnecter, modifier son profil, créer une publication, supprimer ses publications, supprimer son compte, commenter une publication, aimer une publication et commenter un article et contacter la société.
* **Le rédacteur :** il hérite des droits de l’utilisateur. Il peut écrire des articles, les supprimer et les modifier.
* **L’administrateur :** il hérite des droits du rédacteur et il peut supprimer des commentaires, des publications et des articles et bannir ou rendre muet un utilisateur.

Nous avons à présent défini toutes les fonctionnalités que l’application propose et qui y a accès.

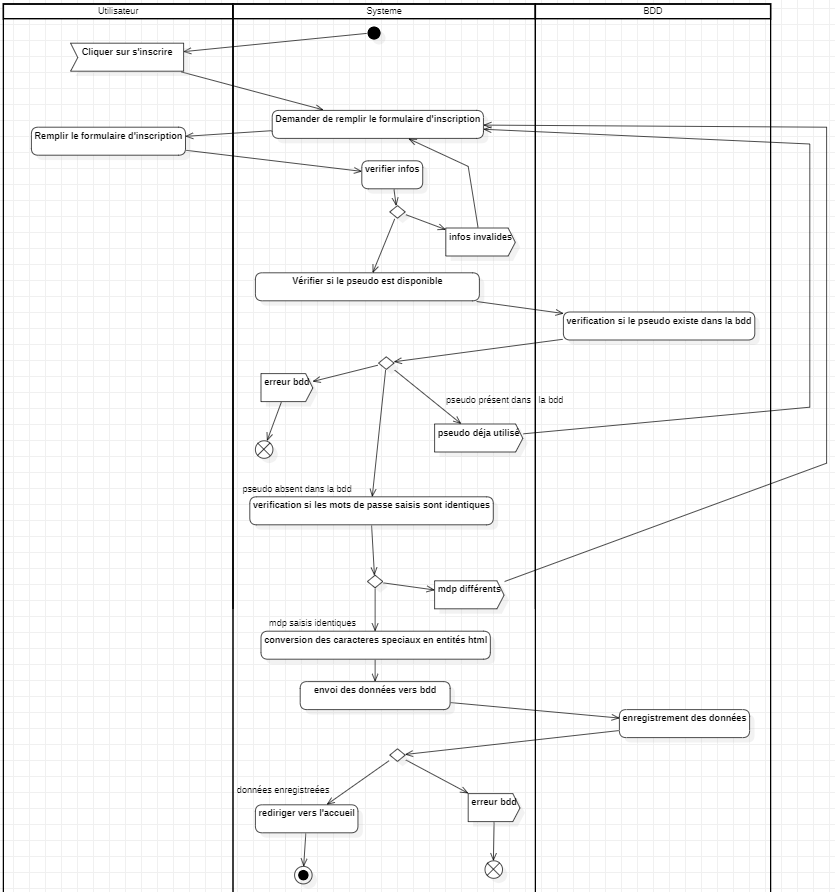
Il est maintenant nécéssaire de détailler le déroulement algorythmique de chacune d’entre elles avec un diagramme d’activité et de séquence.

## Le diagramme d’activité

Le diagramme d’activité est la représentation de la décomposition étape par étape d’une fonctionnalité d’un système d’information il est d’usage d’en créer un par Cas d’utilisation. Il est divisé de manière verticale en plusieurs parties appelées swimlanes selon les acteurs qui interagissent. Chaque swimlane représente un acteur.

Toutes les actions sont reliées par des flèches selon leur ordre et/ou conditions d’exécution.

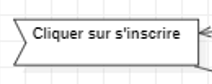
### S’inscrire :



*Etat initial*

### Description de la fonctionnalité

Tout d’abord on reçoit l’action de l’utilisateur, il clique sur « s’inscrire ». Cette action est représentée par un « accept signal ». Un accept signal désigne le moment ou le système d’information reçoit un évènement effectué par un tiers.



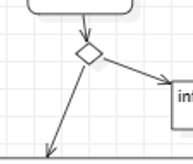
accept signal

Ensuite sera affiché le formulaire d’inscription pour que l’utilisateur le remplisse.

L’utilisateur remplit le formulaire.

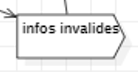
Le système vérifie les informations saisies. C’est-à-dire si tous les champs obligatoires sont remplis.

Après cette action il y a deux suites possibles en fonction de son résultat. Pour représenter cela nous allons utiliser un fork, représenté comme ceci :



fork

* Si les infos sont invalides, il sera envoyé un message d’erreur et il sera redemandé à l’utilisateur de re-remplir le formulaire d’inscription. Le message d’erreur est représenté par un send signal que j’utilise ici indiquer que le système d’information transmet une information à l’utilisateur.



send signal

* Si les informations sont valides, il faut maintenant vérifier si le pseudo choisi par l’utilisateur est disponible. Pour qu’il soit disponible il ne faut pas qu’il soit présent dans la base de données. Il est donc nécessaire d’envoyer une requête à la base de données. Pour qu’elle parcoure les pseudos qui y sont enregistrés.

La base de données exécute la requête. Il y a maintenant trois suites possibles :

* Une erreur de base de données. Il peut y avoir un problème de connexion, le serveur est hors service. Dans ce cas il sera envoyé à l’utilisateur un message d’erreur et ce sera la fin de mon activité. Elle est représentée par un flow final. Le flow final représente une fin de mon activité alors qu’elle n’a pas été entièrement complétée.



flow final

* Si le pseudo est présent dans la base de données, cela veut dire qu’il est déjà utilisé donc pas disponible. On redemande donc à l’utilisateur de remplir le formulaire.
* Si le pseudo n’apparait pas dans la base de données alors il est disponible. L’activité peut donc continuer.

Le système vérifie si le mot de passe choisi et sa confirmation sont identiques.

S’ils sont différents Il est redemandé à l’utilisateur de remplir le formulaire.

S’ils sont identiques, on continue l’activité en convertissant les caractères spéciaux en html pour éviter des injections de code.

Une requête d’insertion est envoyée à la base de données.

la base de données l’exécute.

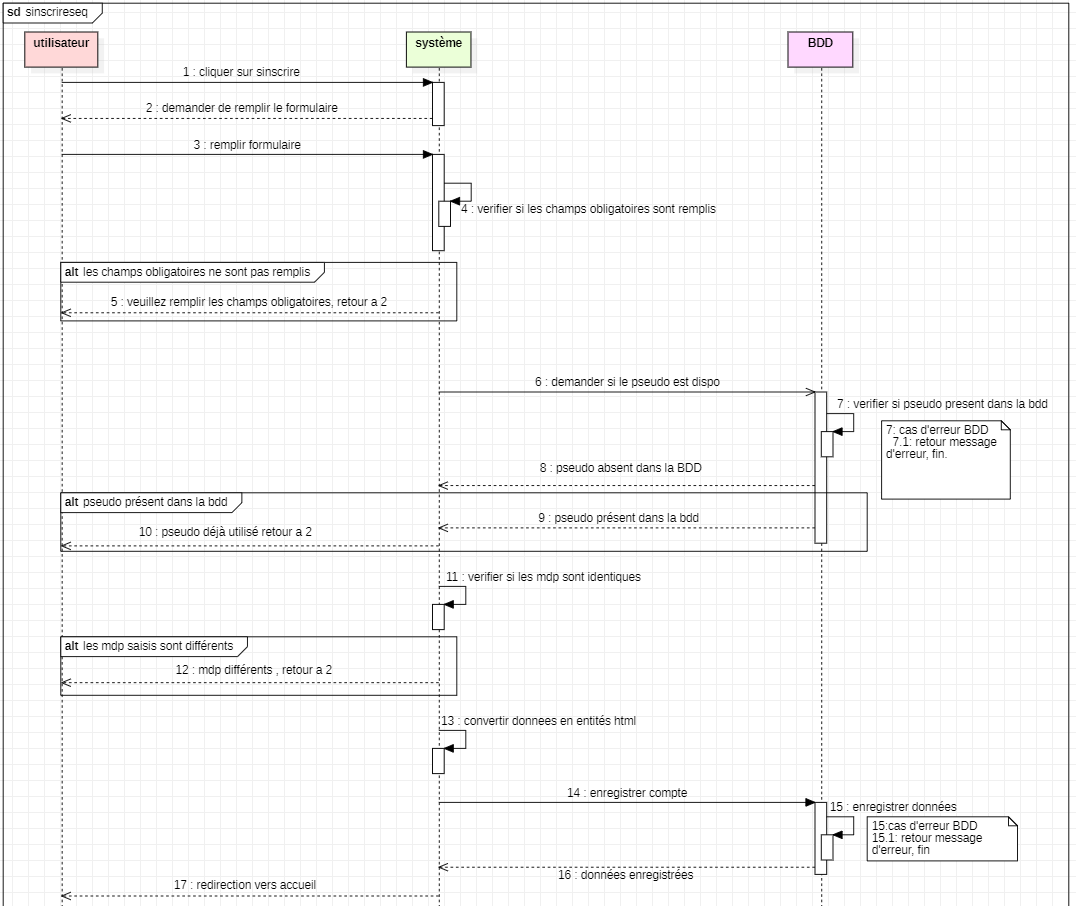
* Cas d’erreur de base de données : on envoie un message d’erreur et on met fin à l’activité par un flow final.
* La requête est exécutée correctement, l’utilisateur est alors redirigé vers l’accueil. L’activité arrive alors à son état final, il s’agit de l’état, contrairement au flow final, où l’activité s’est déroulée correctement.



état final

## Diagramme de séquence

A partir du diagramme d’activité je peux maintenant réaliser un diagramme de séquence pour la même fonctionnalité. Le diagramme de séquence est la représentation graphique des interactions entre le système et les acteurs par ordre chronologique. Cet ordre suit les lignes de vie des acteurs à lire de haut en bas.



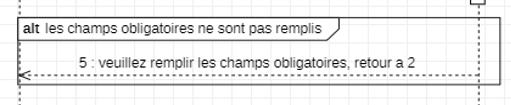
Les messages sont représentés par des flèches. Dans ce diagramme on retrouve deux types de flèches. Des flèches à trait plein et des flèches au trait pointillé. Les flèches au trait plein représentent des demandes d’action et les chèches à trait pointillé représeentent les réponses à ces demandes.

Par exemple, dans le diagramme ci-dessus, à l’étape 1, l’utilisateur clique sur ‘sinscrire c’est donc une demande d’action au système qui, à l’étape 2, répond en affichant le formulaire d’inscription.

Un acteur peut éffectuer un self message. C’est-à-dire exécuter lui-même une tache qu’il lance.Par exemple, à l’étape 4, le système vérifie si tous les champs sont remplis.

Un diagramme de séquence peut représenter des cas alternatifs au cas nominal cela évite de refaire un diagramme pour un déroulement différent. En UML on utilise l’element « «combined fragment » Comme on peut le voir ci-dessus, l’étape 5 traite le cas où les champs obligatoires ne sont pas tous remplis. Ici, le système répond à l’utilisateur en demandant de remplir les champs obligatoires et lui redemande de remplir le formulaire (retour à 2).

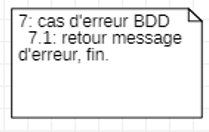
J’ai représenté la condition pour que le scénario alternatif s’exécute dans le titre du combined fragment.



combined fragment, scénario alternatif

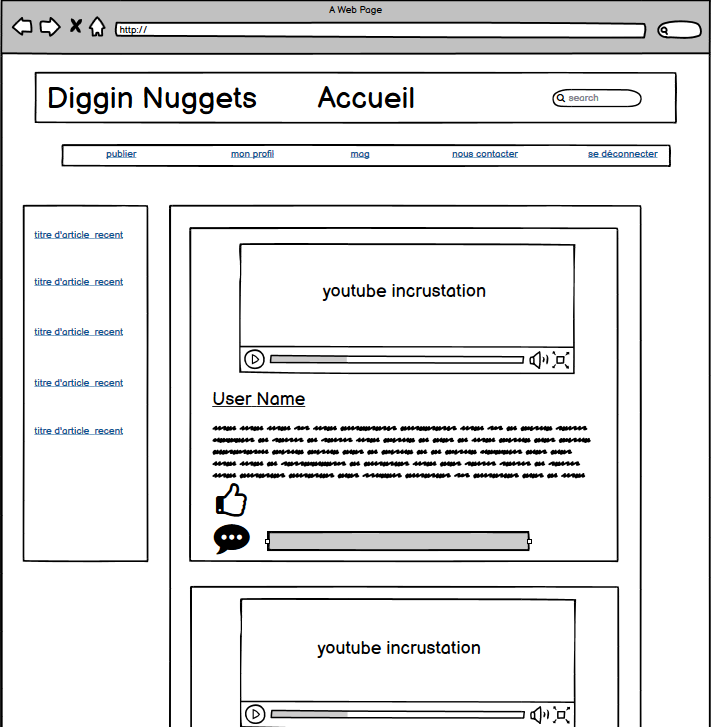
Si la condition n’est pas remplie, on ne prend pas en compte le combined fragment et on continue vers l’étape suivante.

Enfin, cas d’erreur peuvent etre traités avec des notes , par exemple , à l’etape 7, il peut y avoir une erreur de connexion à la base de données pour X raisons. Ici, par exemple la base de données retourne un message d’erreur et on met fin à la séquence.



cas d'erreur

## Maquettage :



Voici la maquette de la page d’accueil de Diggin Nuggets. Elle a été réalisée avec wireframes.

Il s’agit de la page d’accueil pour un utilisateur enregistré et non d’un visiteur, d’un rédacteur ou d’un administrateur.

Depuis cette page , l’utilisateur a accès à la plupart des fonctionnalités qui lui sont autorisées dans le diagramme de cas d’utilisation. Il peut , depuis l’accueil, voir les publications sur le fil d’actualité, accéder au magasine pour lire des articles, contacter l’entreprise , faire une recherche en fonction du style de publication qu’il veut voir, accéder au formulaire de publication et , dans la section à gauche des articles récemment publiés lui sont proposés.

Les publications sont affichées sur le fil d’actualité de l’accueil, de la plus récente à la plus ancienne. L’utilisateur peut « aimer » et commenter chacune des publications.

Chaque publication est composée d’une incrustation youtube, du pseudo de l’utilisateur qui l’a créée, le pseudo redirige vers le profil de cet utilisateur oè l’on peut voir toutes les publications qu’il a posté.

Il est nécéssaire que l’utilisateur arrive à chacune des fonctionnalités en maximum trois clics.

## Arborescence :

Dans Diggin Nuggets, l’utilisateur n’est qu’à deux clics de chaque fonctionnalité en partant de l’accueil. En effet, depuis l’accueil, il peut visualiser les publications, et à partir de là, visiter le profil de l’auteur. Toujours depuis l’accueil, il peut accéder aux pages Publier, Mon profil, Contact et Mag, où il retrouvera tous les articles. Il peut également faire une recherche dans la barre de recherche, avoir accès à des articles récents et se déconnecter.



Ce schéma d’arborescence a été réalisé avec le site gratuit Gloomaps. Chaque étage représente un clic en partant de l’accueil.

# **CONCEPTION**

Une fois les besoins et fonctionnalités spécifiées il faut définir la structure de la base des données. Pour cela, j’utilise la méthode d’analyse Merise pour définir un Modèle Conceptuel de Données (MCD) qui me permettra de construire un Modèle Logique de Données (MLD). C’est à partir de ces diagrammes que je pourrai enfin construire la base de données du site.

## Modèle Conceptuel de Données (MCD)

(Voir diagramme à la page suivante)

Le MCD sert tout d’abord à lister et organiser les données qu’il faut stocker dans la base de données. Il est composé d’entités qui sont elles-mêmes composées d’attributs.

Chaque attribut représente une caractéristique de l’entité et il peut avoir plusieurs types comme par exemple VARCHAR pour une chaine de caractères, on peut aussi choisir le nombre de caractères, sur JMerise la valeur par défaut pour VARCHAR est de 50 caractères. Il y a aussi les données de type INT qui représentent un nombre entier (integer), ou encore DATE pour une date et DATETIME pour une date et une heure . Le type INT peut être décliné en TINYINT ou en BIGINT selon le besoin, TINYINT pouvant représenter des valeurs allant de 0 à 255 (ce sont les valeurs que peut représenter un octet) et BIGINT de -2^63 (-9 223 372 036 854 775 808) à 2^63-1 (9 223 372 036 854 775 807) ce sont toutes les valeurs pouvant être représentées par huit octets.

Chaque entité dispose d’un identifiant. Il s’agit du premier attribut représenté dans l’entité. Il permettra de pouvoir une enregistrement. Sa valeur est donc unique et de type INT.

Ici, pour l’entité utilisateurs, le premier attribut est id\_utilisateur de type INT. On retrouve ensuite le pseudo, le mail, le nom, le prénom et le mot de passe. Ces données devront être de type VARCHAR car elles sont des chaines de caractères.

Ces attributs sont suivis par l’âge de l’utilisateur. J’ai décidé de le typer en TINYINT car jusqu’à preuve du contraire, les humains ne peuvent pas vivre plus de 255 ans.Il n’est donc pas décéssaire de stocker de valeur supérieure à ce chiffre qui est la valeur maximale pouvant être représentée par un TINYINT.

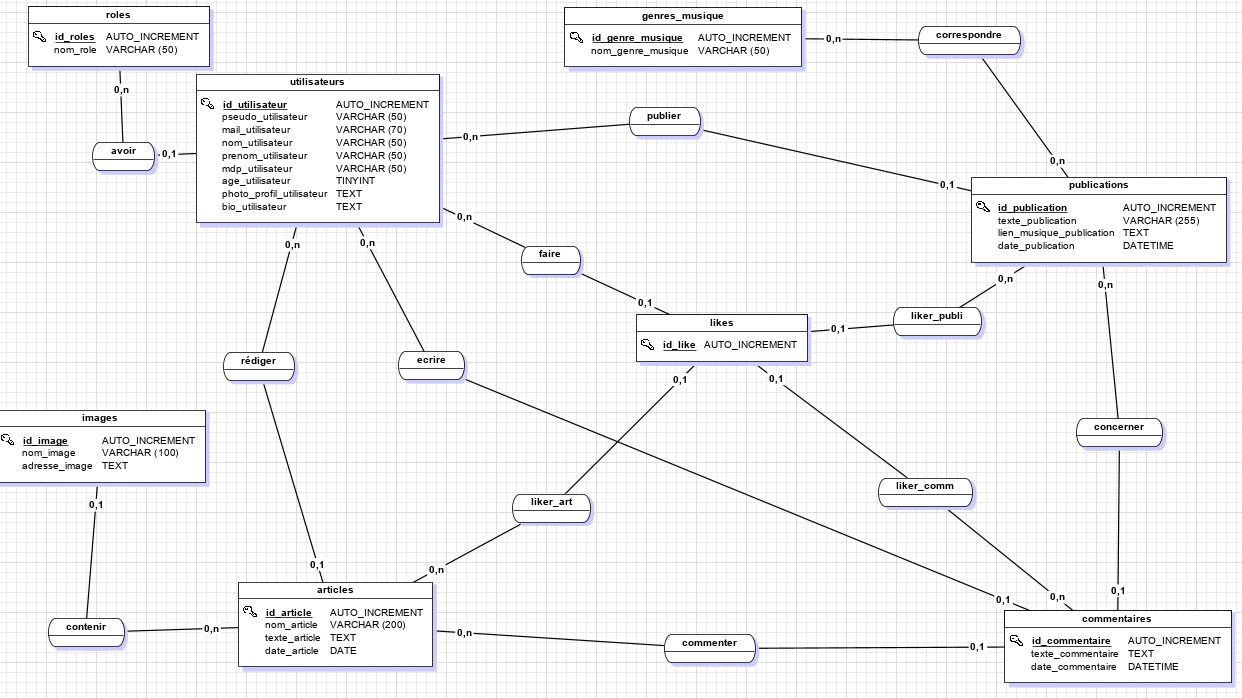
Toutes ces entités sont associées entre elles. Chaque association a un nom unique défini par un verbe à l’infinitif et est définie par sa cardinalité. La cardinalité est définie dans les deux sens de l’association et elle représente le nombre minimum et le nombre maximum d’instances que l’on peut avoir pour chaque association.

Dans mon diagramme, regardons l’association PUBLIER , entre les entités utilisateurs et publications :

Sur Diggin nuggets, un utilisateur peut PUBLIER de zéro à plusieurs publications. Du coté utilisateurs de l’association on représentera ceci par « 0,n » (0 pour le minimum , n pour le maximum).Une publication peut etre publiée par au maximum un seul utilisateur et au minimum par zéro. Il semblerait plus logique de dire qu’elle doit être publiée par au moins un utilisateur mais je voulais éviter les suppressions en cascade pour ce projet. Du côté publications de l’association, la cardinalité est de 0,1 (une publication peut être publiée par 0 à un utilisateur).

Regardons maintenant l’association CORRESPONDRE , entre genres\_musique et publications :

Une publication peut CORRESPONDRE avec de 0 à plusieurs genres donc du coté publications on a une cardinalité de « 0,n ». Un genre de musique peut correspondre avec de 0 à plusieurs publications donc de ce côté aussi, la cardinalité est de « 0,n ».



Modèle Conceptuel de Données de Diggin Nuggets.

## Modèle Logique de Données (MLD)

(voir digramme à la page suivante)

A partir du MCD que j’ai réalisé, j’ai pu réaliser le MLD correspondant. Le MLD permet de donner les instructions finales avant la création de la base de données.

Dans le MLD, les entités deviennent des tables et les associations deviennent des relations et l’identifiant devient clé primaire. La clé primaire permet d’identifier un enregistrement dans une base de données.

Reprennons les associations que nous avons décrit précédemment sur le MCD :

L’association PUBLIER est de 0,n du coté de l’utilisateur car il peut publier de zéro à plusieurs publications et du coté de publication elle est de 0,n car une publication peu etre publiée par entre zéro et plusieurs utilisateurs.

Cette association va devenir une relation de type 0.1 de publications vers utilisateurs et est représentée avec une flèche. Une publication ne peut avoir qu’un utilisateur maximum. La table publications absorbe alors la clé primaire « id\_utilisateur » d’utilisateurs et en fait une clé étrangère.

Maintenant, regardons l’association CORRESPONDRE entre genres\_musique et publications : c’est une association 0,n - 0,n. on dit que c’est une relation de type N. En MLD, il est nécessaire de créer une table d’association qui absorbe les clés primaires des tables de chaque côté et deviennent clés étrangères. Il faut donc créer la table d’association correspondre qui a deux clés primaires qui sont elles-mêmes des clés étrangères.

## 

Modèle Logique de données de Diggin Nuggets.

# **OUTILS UTILISES**

## PHP logo PNGPhp en modèle mvc

## Formation HTML / CSS débutant | Numgrade l'expert reconnu

## HTML et CSS

## Mysql Logo -Logo Brands For Free HD 3D MySQL

## MCD_MLD | Facebook JMERISE

# BalsamiqBALSAMIQ wirframes

# Download XAMPP 32/64bit For Windows 7/8/8.1/10 - Andri Tekno | News ...Xampp

# StarUMLstar UML

# Download dan install Git - Tech BodyFitStation Git

# **FONCTIONNALITES**

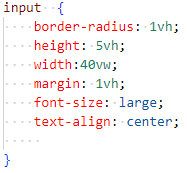
Ce projet est réalisé PHP modèle MVC (Model-Vue-Controller) c’est-à-dire que mon code est divisé en trois types de fichiers :

* Le modèle : il s’agit d’une classe , il contient tous les outils nécessaires à l’instanciation d’un objet
* La vue : elle traite l’affichage de la page.
* Le controller : c’est le fichier qui traite l’algorithmie de la fonctionnalité. Il fait appel à la vue et au modèle.

## Inscription :

Voici la page en version portable :

La largeur des conteneurs s’adapte en fonction de la taille de l’écran. Pour ce faire j’ai utilisé des unités de mesure CSS relatives à la taille de l’ecran.

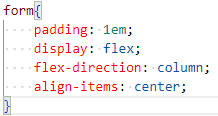


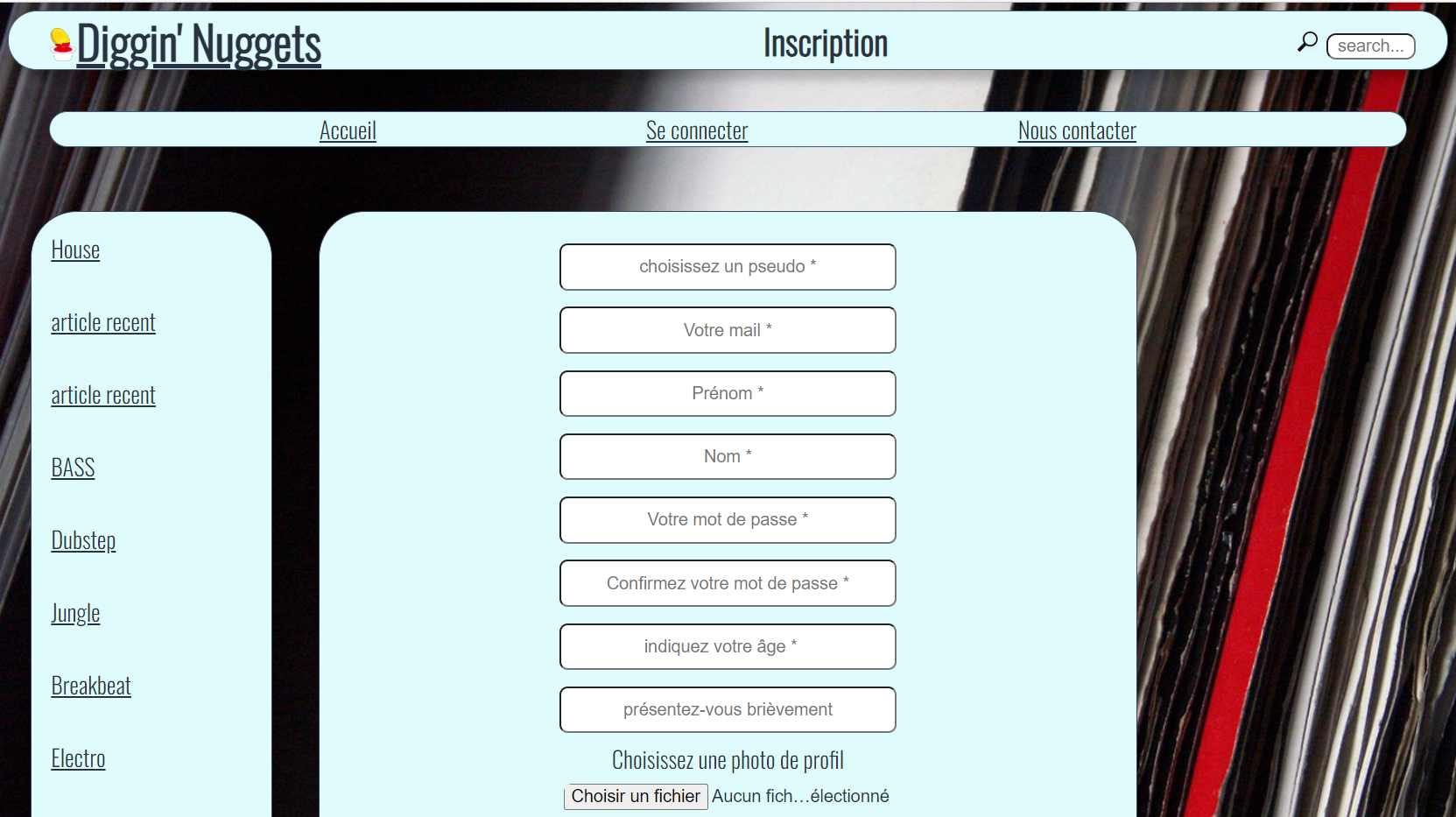
Les vh (view height) s’adaptent à la hauteur de l’écran et les vw(view width) à sa largeur.



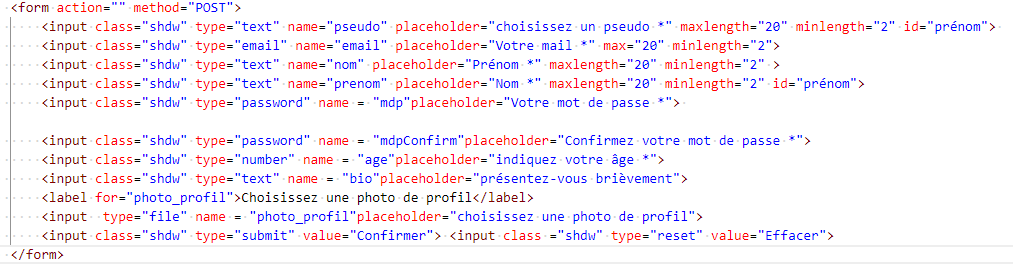
La voici en version écran d’ordinateur :

Pour la disposition du formulaire j’ai utilisé l’outil flexbox pour que les éléments qui le composent soient disposés en une colonne centrée horizontalement au sein de son conteneur.





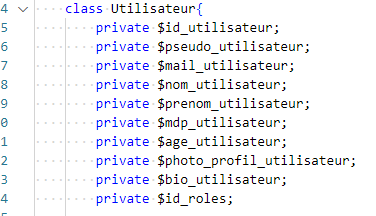
Voici le code html du formulaire d’inscription :



Il est en méthode post car il sert à insérer des données en base de données et non à récupérer des données. Grace à l’attribut « type » je définis le type de chaque saisie. Ici nous pouvons voir les types email, text, password, number et file (pour des fichiers). L’attribut name sert à identifier chaque élément dans le controleur php. Chaque élément sera identifiable avec la superglobale $\_POST.

Si je veux par exemple récupérer ce qu’a saisi l’utilisateur dans le champ « pseudo » je devrai faire .

### Classe utilisateur (Model) :



Je mets tous les attributs en private de manière à ce que je ne puisse y avoir accès de manière directe seulement à l’intérieur de la classe. Pour les manipuler depuis le controlleur je devrai utiliser les getters et les setters qui sont des fonctions publiques , donc accessibles depuis le controlleur.

Voici un getter qui retourne l’id d’un utilisateur :

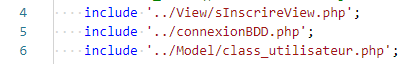


Voici un setter , qui permet de définir le pseudo d’un utilisateur :

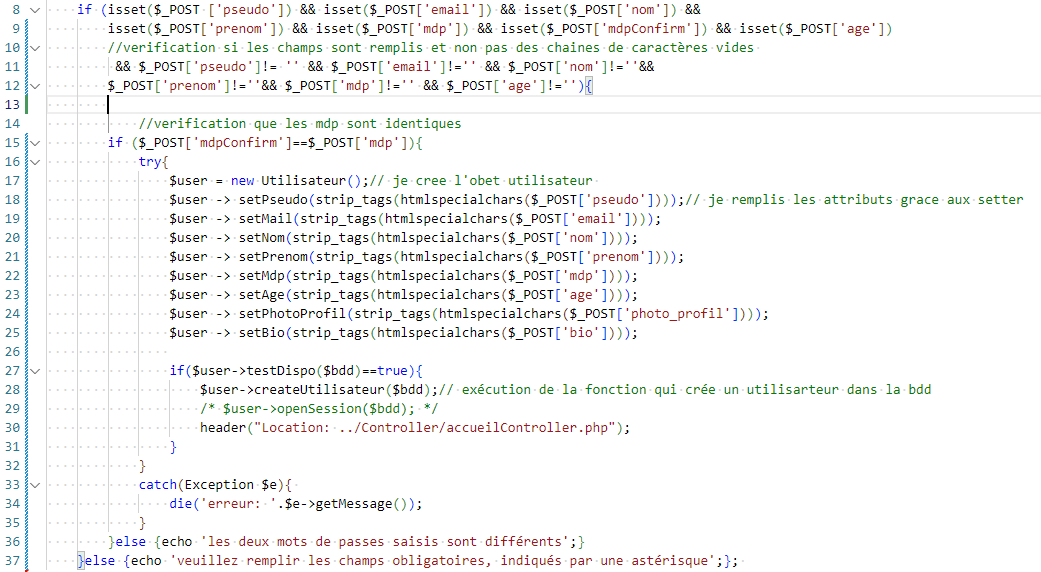


Je dispose d’un getter et d’un setter pour chaque attribut de ma classe.

### Dans mon Controlleur :

En premier lieu, je fais appel mes fichiers vue, model et ma connexion à la base de données. 

Ensuite je vérifie avec isset si les les variables $\_POST sont déclarées et non nulles. En testant mon code j’ai pu remarquer que dans certains cas, php pouvait une variable contenant une chaine de caractère vide comme non nulle, alors je m’assure à partir de la ligne 11 que mes variables ne sont pas des chaines de caractères vides. Si ces conditions sont remplies, je passe à la suite en vérifiant que le mot de passe choisi est identique à sa confirmation.



Si les mots de passe correspondent , j’instancie un nouvel objet utilisateur et lui attribue les données saisies grâce aux setter en prenant soin d’éviter les injections de code grâce aux fonctionnalités strip\_tags et htmlspecialchars. Strip\_tags supprime les balises HTML et PHP d’une chaîne de caractères et htmlspecialchars convertit les caractères spéciaux en entités html.

Je vérifie ensuite si le pseudo est disponible et qu’il n’existe pas déjà un compte utilisant l’adresse mail saisie grâce à la méthode testDispo que je vais détailler plus bas. Si elle retourne true le pseudo est disponible et l’adresse mail n’est pas utilisée, on peut donc continuer en lançant la fonction createUtilisateur qui est aussi expliquée dans les pages suivantes. Après cela, on redirige l’utilisateur vers l’accueil, ce qui est l’état final de la fonctionnalité.

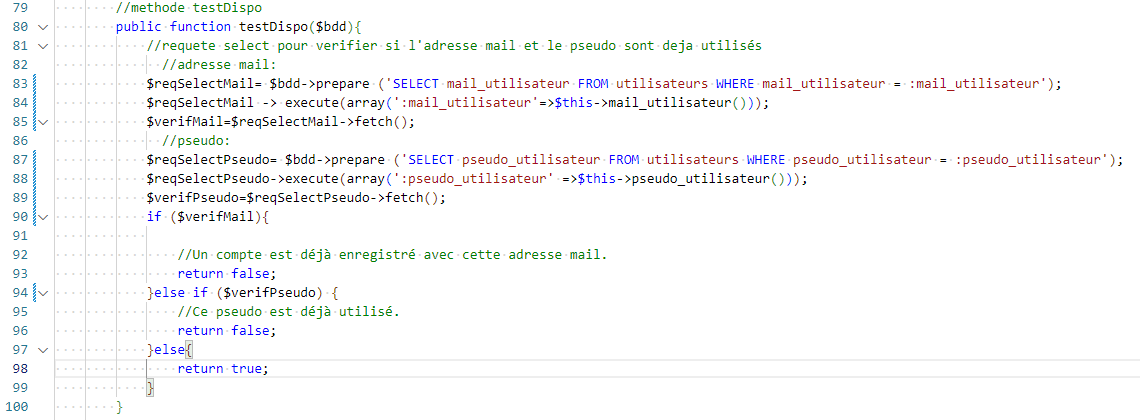
### Dans le model :

Méthode testDispo :

Ici je cherche à vérifier si le pseudo saisi par l’utilisateur est disponible et que l’adresse mail saisie n’est pas utilisée par un compte existant. Pour cela je prépare d’abord une requête SQL select pour récupérer les mails qui sont identiques a celui saisi pas l’utilisateur. Je l’exécute puis je stocke dans la variable $dispoMail ce que la requête retourne. Je fais la meme opération pour le pseudo.et stocke le retour de la requête dans $dispoPseudo .

Si $verifMail ou $verifPseudo est true cela veut dire que le mail ou le pseudo est présent dans la BDD donc qu’il est déjà utilisé . Il n’est alors pas disponible. Dans ce cas la fonction testDispo retourne false.

Si $verifMail et $verifpseudo ne retournent rien, cela veut dire que le mail et le pseudo saisis ne sont pas présents dans la BDD, ils sont donc disponibles. Dans ce cas la fonction testDispo retourne true



Méthode createUtilisateur :

Cette méthode sert à enregistrer les données d’un utilisateur dans la base de données.



Je prépare une requête SQL de type insert que je stocke dans la variable reqInsert . Les valeurs ne sont pas encore attribuées. J’exécute la requête et à ce moment là, je donne des valeurs aux paramètres grâce aux données instanciées dans mon objet.

# **CONCLUSION**

Le site web n’est pas encore finalisé. En effet, il me reste une très grande partie du travail à effectuer comme l’affichage des publications sur le fil d’actualité, la partie magazine, la section commentaires ou encore la gestion des mentions « j’aime ».

Le projet m’a donné et continue de me donner beaucoup de fil à retordre. J’ai dû le recommencer la conception et le développement et l’analyse fonctionnelle plusieurs fois. Même si cela m’a fait prendre du retard sur le projet, j’en ai appris beaucoup sur l’organisation et l’anticipation de certaines situations. Il m’est par exemple arrivé, en plein développement, pensant avoir finalisé la conception, de me rendre compte que la structure de ma base de données m’empêchait de développer correctement les mentions j’aime et les commentaires.

Grace à d’innombrables erreurs de ce type je commence à avoir un peu plus de facilités à visualiser un projet dans son ensemble et adopter une bonne méthodologie de travail.

Je compte continuer de travailler sur la partie non finalisée du site et le finir pendant mon temps libre pour continuer de progresser après la fin de la formation. Cela pourrait également me permettre d’avoir un projet à présenter à de potentiels recruteurs.

# **ANNEXES**

## SQL :

CREATE DATABASE diggin\_nuggets;

use diggin\_nuggets;

CREATE TABLE `articles` (

  `id\_article` int(11) NOT NULL,

  `nom\_article` varchar(200) NOT NULL,

  `texte\_article` text NOT NULL,

  `date\_article` date DEFAULT NULL,

  `id\_utilisateur` int(11) DEFAULT NULL

)

CREATE TABLE `commentaires` (

  `id\_commentaire` int(11) NOT NULL,

  `texte\_commentaire` text NOT NULL,

  `date\_commentaire` datetime DEFAULT NULL,

  `id\_utilisateur` int(11) DEFAULT NULL,

  `id\_publication` int(11) DEFAULT NULL,

  `id\_article` int(11) DEFAULT NULL

)

CREATE TABLE `correspondre` (

  `id\_genre\_musique` int(11) NOT NULL,

  `id\_publication` int(11) NOT NULL

)

CREATE TABLE `genres\_musique` (

  `id\_genre\_musique` int(11) NOT NULL,

  `nom\_genre\_musique` varchar(50) DEFAULT NULL

)

CREATE TABLE `images` (

  `id\_image` int(11) NOT NULL,

  `nom\_image` varchar(100) DEFAULT NULL,

  `adresse\_image` text NOT NULL,

  `id\_article` int(11) DEFAULT NULL

)

CREATE TABLE `likes` (

  `id\_like` int(11) NOT NULL,

  `id\_utilisateur` int(11) NOT NULL,

  `id\_publication` int(11) DEFAULT NULL,

  `id\_commentaire` int(11) DEFAULT NULL,

  `id\_article` int(11) DEFAULT NULL

)

CREATE TABLE `publications` (

  `id\_publication` int(11) NOT NULL,

  `texte\_publication` varchar(255) NOT NULL,

  `lien\_musique\_publication` text NOT NULL,

  `date\_publication` datetime DEFAULT NULL,

  `id\_utilisateur` int(11) DEFAULT NULL

)

CREATE TABLE `roles` (

  `id\_roles` int(11) NOT NULL,

  `nom\_role` varchar(50) DEFAULT NULL

)

CREATE TABLE `utilisateurs` (

  `id\_utilisateur` int(11) NOT NULL,

  `pseudo\_utilisateur` varchar(50) NOT NULL,

  `mail\_utilisateur` varchar(70) NOT NULL,

  `nom\_utilisateur` varchar(50) NOT NULL,

  `prenom\_utilisateur` varchar(50) NOT NULL,

  `age\_utilisateur` tinyint(4) NOT NULL,

  `photo\_profil\_utilisateur` text DEFAULT NULL,

  `bio\_utilisateur` text DEFAULT NULL,

  `id\_roles` int(11) DEFAULT NULL,

  `mdp\_utilisateur` varchar(100) DEFAULT NULL

)

ALTER TABLE `articles`

  ADD PRIMARY KEY (`id\_article`);

ALTER TABLE `commentaires`

  ADD PRIMARY KEY (`id\_commentaire`),

  ADD KEY `id\_utilisateur` (`id\_utilisateur`),

  ADD KEY `id\_publication` (`id\_publication`),

  ADD KEY `id\_article` (`id\_article`);

ALTER TABLE `correspondre`

  ADD PRIMARY KEY (`id\_genre\_musique`,`id\_publication`),

  ADD KEY `id\_publication` (`id\_publication`);

ALTER TABLE `genres\_musique`

  ADD PRIMARY KEY (`id\_genre\_musique`);

ALTER TABLE `images`

  ADD PRIMARY KEY (`id\_image`),

  ADD KEY `id\_article` (`id\_article`);

ALTER TABLE `likes`

  ADD PRIMARY KEY (`id\_like`),

  ADD KEY `id\_utilisateur` (`id\_utilisateur`),

  ADD KEY `id\_publication` (`id\_publication`),

  ADD KEY `id\_commentaire` (`id\_commentaire`),

  ADD KEY `id\_article` (`id\_article`);

ALTER TABLE `publications`

  ADD PRIMARY KEY (`id\_publication`),

  ADD KEY `id\_utilisateur` (`id\_utilisateur`);

ALTER TABLE `roles`

  ADD PRIMARY KEY (`id\_roles`);

ALTER TABLE `utilisateurs`

  ADD PRIMARY KEY (`id\_utilisateur`),

  ADD KEY `id\_roles` (`id\_roles`);

ALTER TABLE `articles`

  MODIFY `id\_article` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE `commentaires`

  MODIFY `id\_commentaire` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE `genres\_musique`

  MODIFY `id\_genre\_musique` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE `images`

  MODIFY `id\_image` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE `likes`

  MODIFY `id\_like` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE `publications`

  MODIFY `id\_publication` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE `roles`

  MODIFY `id\_roles` int(11) NOT NULL AUTO\_INCREMENT;

ALTER TABLE `utilisateurs`

  MODIFY `id\_utilisateur` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=92;

ALTER TABLE `commentaires`

  ADD CONSTRAINT `commentaires\_ibfk\_1` FOREIGN KEY (`id\_utilisateur`) REFERENCES `utilisateurs` (`id\_utilisateur`),

  ADD CONSTRAINT `commentaires\_ibfk\_2` FOREIGN KEY (`id\_publication`) REFERENCES `publications` (`id\_publication`),

  ADD CONSTRAINT `commentaires\_ibfk\_3` FOREIGN KEY (`id\_article`) REFERENCES `articles` (`id\_article`);

ALTER TABLE `correspondre`

  ADD CONSTRAINT `correspondre\_ibfk\_1` FOREIGN KEY (`id\_genre\_musique`) REFERENCES `genres\_musique` (`id\_genre\_musique`),

  ADD CONSTRAINT `correspondre\_ibfk\_2` FOREIGN KEY (`id\_publication`) REFERENCES `publications` (`id\_publication`);

ALTER TABLE `images`

  ADD CONSTRAINT `images\_ibfk\_1` FOREIGN KEY (`id\_article`) REFERENCES `articles` (`id\_article`);

ALTER TABLE `likes`

  ADD CONSTRAINT `likes\_ibfk\_1` FOREIGN KEY (`id\_utilisateur`) REFERENCES `utilisateurs` (`id\_utilisateur`),

  ADD CONSTRAINT `likes\_ibfk\_2` FOREIGN KEY (`id\_publication`) REFERENCES `publications` (`id\_publication`),

  ADD CONSTRAINT `likes\_ibfk\_3` FOREIGN KEY (`id\_commentaire`) REFERENCES `commentaires` (`id\_commentaire`),

  ADD CONSTRAINT `likes\_ibfk\_4` FOREIGN KEY (`id\_article`) REFERENCES `articles` (`id\_article`),

  ADD CONSTRAINT `likes\_ibfk\_5` FOREIGN KEY (`id\_utilisateur`) REFERENCES `utilisateurs` (`id\_utilisateur`),

  ADD CONSTRAINT `likes\_ibfk\_6` FOREIGN KEY (`id\_publication`) REFERENCES `publications` (`id\_publication`),

  ADD CONSTRAINT `likes\_ibfk\_7` FOREIGN KEY (`id\_commentaire`) REFERENCES `commentaires` (`id\_commentaire`),

  ADD CONSTRAINT `likes\_ibfk\_8` FOREIGN KEY (`id\_article`) REFERENCES `articles` (`id\_article`);

ALTER TABLE `publications`

  ADD CONSTRAINT `publications\_ibfk\_1` FOREIGN KEY (`id\_utilisateur`) REFERENCES `utilisateurs` (`id\_utilisateur`),

  ADD CONSTRAINT `publications\_ibfk\_2` FOREIGN KEY (`id\_utilisateur`) REFERENCES `utilisateurs` (`id\_utilisateur`),

  ADD CONSTRAINT `publications\_ibfk\_3` FOREIGN KEY (`id\_utilisateur`) REFERENCES `utilisateurs` (`id\_utilisateur`);

ALTER TABLE `utilisateurs`

  ADD CONSTRAINT `utilisateurs\_ibfk\_1` FOREIGN KEY (`id\_roles`) REFERENCES `roles` (`id\_roles`);