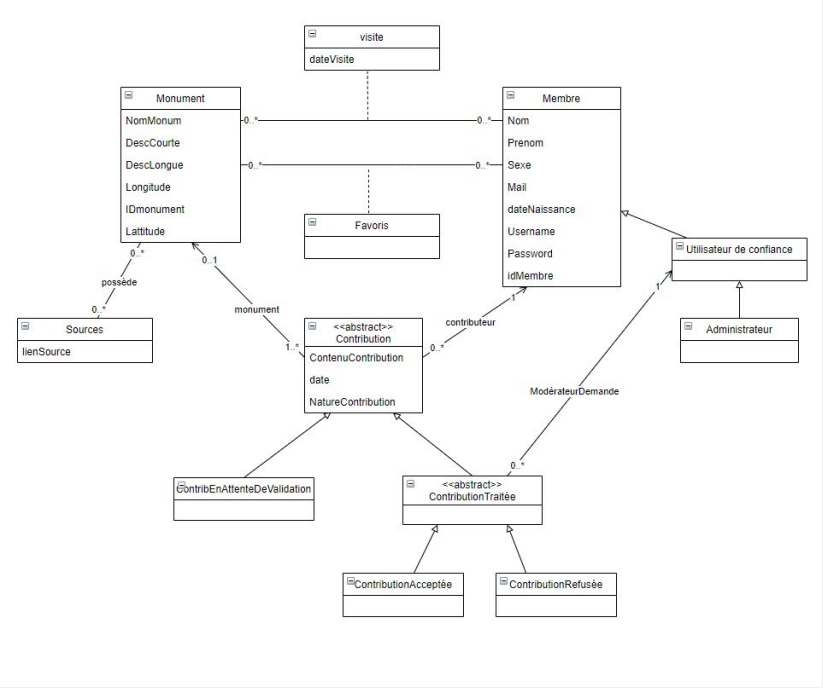
Travail sur la conception de la base de données

Schéma de base de donnée réalisé avant l'itération est présent dans l'étude préalable :



On en déduit une proposition de modèle relationnel en utilisant la méthode qui consiste à réunir les classes héritant d'autre en une seule avec des attributs supplémentaire :

Membre : (idMembre, Nom, Prenom, Sexe, Mail, dateNaiss, Username, Password, rôle)

rôle ∈ { Membre, utilisateurDeConfiance, Administrateur}

Monument : (IDMonument, NomMonum, DescCourte, DescLongue, Longitude, Lattitude)

Source: (lienSource, #IDMonument)

Contribution : (idContribution, #idMonument, #idMembre, ContenuContribution, date, statutContribution, contenuContribution, idMembreModerateur)

statutContribution ∈ { EnAttenteDeValidation, Acceptée, Refusée }

natureContribution ∈ { CréationMonument, ModificationMonument }

Favoris : (#idMonument, #membre)

Visite : (#idMonument, #membre, dateVisite)

Prise en compte du RDV avec monsieur Guénego

Il est ressorti de ce RDV plusieurs possibles modifications à apporter à notre future base de données :

- Un utilisateur doit pouvoir accéder à l'ensemble des fonctionnalités sans avoir créé de compte :
    - Pour cela on peut se servir techniquement d'un système de cookies.
    - Dans la base cela peut se traduire par un membre qui aura un rôle "NonInscrit".

Système de rôle

Jusqu'ici le système de rôle se résumait à un attribut dans la table Membre.
Il me paraît intéressant de créer un table rôle à part pour deux raisons :

- Cela permet d'associer un nombre entier à chaque rôle et ainsi éviter de manipuler des chaînes de caractère comme des constantes.
- On peut ainsi définir un certain nombre d'attribut boolean qui correspondent aux privilèges du rôle

On a donc :
Role : (idRole, nomRole, permDeBase, permModererContrib, permProposerContrib, permAdministration)
rôle ∈ { Membre, utilisateurDeConfiance, Administrateur, nonInscritAvecCookie, nonInscritSansCookie}
et :
Membre : (idUser, Nom, Prenom, Sexe, Mail, dateNaiss, Username, Password, #role)
si role = nonInscrit alors tous sauf idUser est null

avec :

- permModererContrib : Permissions permettant d'accepter ou de refuser les propositions de contributions des membres
- permProposerContrib : Permissions permettant de proposer des contributions (pour rappel cela peut être un nouveau monument, ou une modification sur un monument existant)
- permAdministration : Ensemble de privilège d'administration permettant entre autre de supprimer ou modifier n'importe quel membre, d'ajouter ou de modifier n'importe quel monument
- permEnregistrement : Ensemble de privilèges permettant d'enregistrer des favoris ou des listes ou encore d'avoir accès à un historique des visites. A priori les seuls utilisateurs qui ne le posséderont pas seront ceux qui n'ont pas de compte et refusent les cookies.
- Tous les utilisateurs peuvent accéder à l'application et consulter leurs emplacements et les monuments sur la carte entre autres, il n'y a donc pas de permissions associées à ces fonctionnalités de base.

Cette liste de permission est sujette à modification si le besoin s'en fait ressentir.

Système de Membre

Les membres sont définis par un certains nombre d'attributs :

Membre : (idUser, Nom, Prenom, Sexe, Mail, dateNaiss, Username, Password, #role)

Le sexe ne pourra avoir qu'un certain nombre de valeur (homme, femme, nonCommuniqué et autre) par exemple. Il est donc adapté d'utiliser un attribut de type *ENUM* qui convertira automatiquement la chaîne de caractère en nombre entier afin d'éviter de stocker des chaînes de caractères inutilement. Cet attribut se déclare comme suit :

```
`sexe` ENUM('homme','femme','autre','non-renseigné') NOT NULL DEFAULT
'non-renseigné',
```

Système de liste de monuments

Au sein du groupe nous avons pensé à un système de liste de monument (comparable à des playlists musicales) qui permettrait à un utilisateur de créer une liste (privée ou public) avec un titre, une description et une liste ordonnée de monuments.
Cela se traduit par les tables :

ListeMonument : (idListe, nom, description, #créateur, visibilité, dateCréation
visibilité ∈ { MoiUniquement, MembreDisposantDuLien, Public }
AppartenanceListe : (#idListe, #idMonument, numeroDansListe)

La visibilité ne pourra avoir qu'un certain nombre de valeurs (MoiUniquement, MembreDisposantDuLien, Public). Il est donc adapté d'utiliser un attribut de type *ENUM* pour les mêmes raisons que vu précédemment. Cet attribut se déclare ainsi :

```
`visibilite` ENUM('toutLeMonde','moiUniquement','UtilisateurAvecLien') NOT
NULL DEFAULT 'moiUniquement' ,
```

Gestion des images

Les images ne seront pas dans la base de données (pour cela une base NOSQL dédiée serait plus adaptée). Mais stockées sur le serveur de l'application et référencées à partir d'un id et d'un chemin quand c'est nécessaire dans la base. L'id suffira à retrouver le chemin du fichier de l'image dans notre application PHP.

Système de contribution

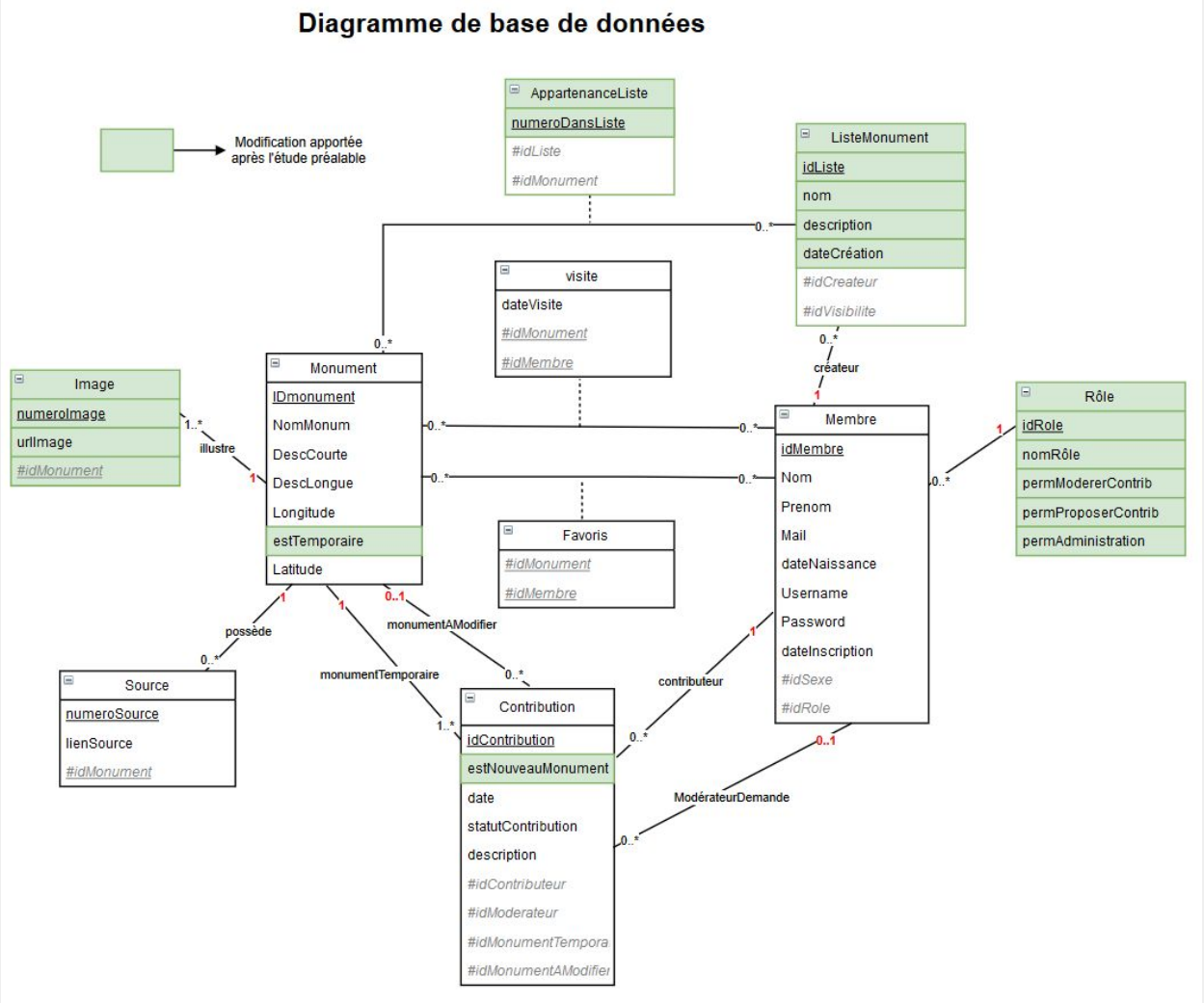
Les contributions correspondent à une proposition de création ou modification de monument. Cela est symbolisé par un attribut boolean *estNouveauMonument.*

Une contribution peut avoir plusieurs statuts : en attente de traitement, acceptée, refusée. Pour cela on définit un attribut entier qui aura respectivement comme valeur 1, 2 ou 3.

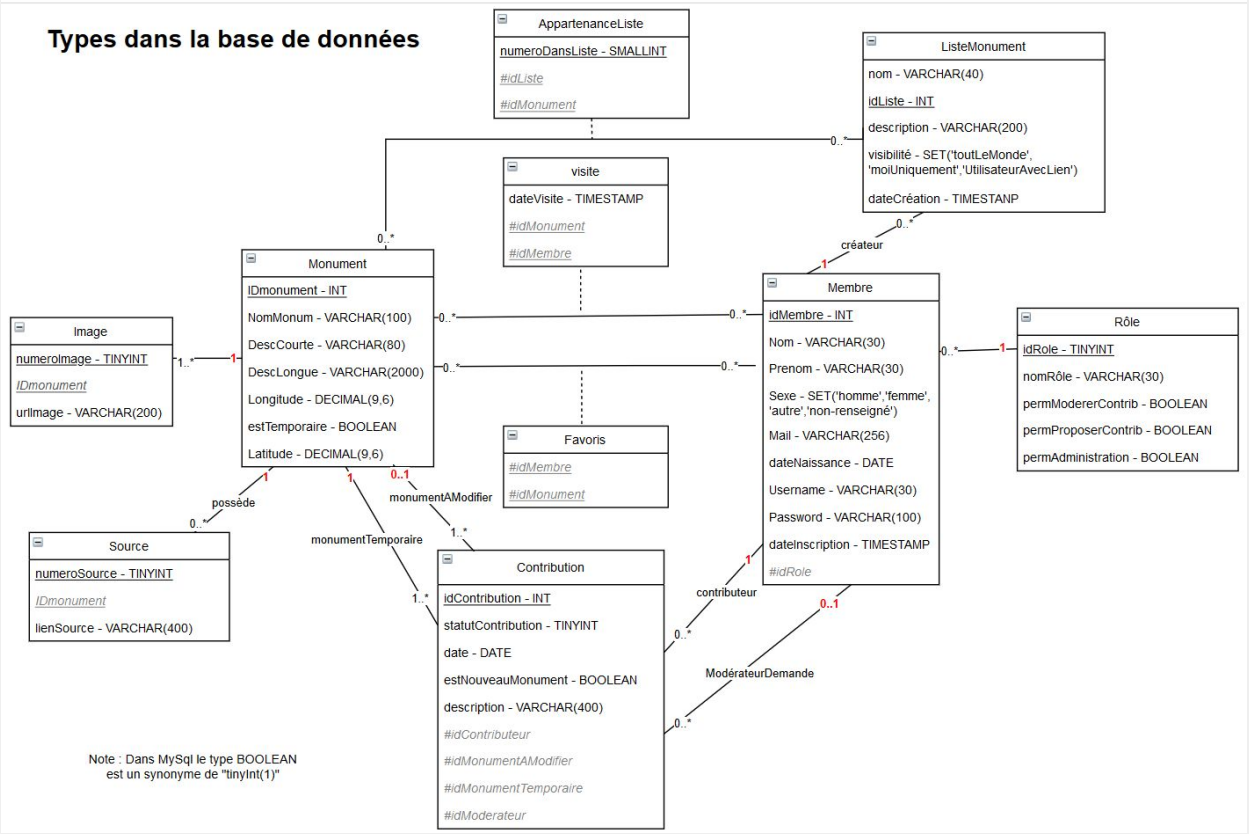
Le contenu de la contribution en elle-même est représenté par un monument qui a son attribut *estTemporaireMonument* a vrai. Si un nouveau monument est accepté, alors il devient automatiquement un monument valide. Si c'est une modification qui est acceptée alors le monument initial est supprimé et le monument temporaire prend son id.

Un attribut de type varchar *description* sert au contributeur à synthétiser les modifications qu'il a apportées. Dans le cas de la création d'un monument sa valeur sera null.

Diagramme de synthèse :



Choix des types mysql :



Précisions :

- Tous les attributs de type *int* et *tinyint* et *smallint* (donc les id) sont *UNSIGNED* car il n'y aura pas d'id négatif.
- Les coordonnées sont stockées en degré avec des nombres décimaux de forme xxx,xxxxxx.
- D'après plusieurs sources la longueur maximale d'une adresse email est 256 caractères, c'est donc un varchar(256) que nous utilisons dans la base.

Création de la base mysql

Instructions pour la création des tables :

Voir fichier /Iteration_1/Documents/Base de données/creation_table.sql

Clés étrangères

Voir fichier /Iteration_1/Documents/Base de données/foreign_key.sql

Ajout des index

Les index servent à l'optimisation des recherches dans la base, ainsi qu'à la définition de contraintes (le plus souvent uniques) sur un ou plusieurs attributs.
Les index correspondant aux clés primaires et étrangères ont déjà été définis lors de la création des tables.

Voir les instructions pour les créations d'index : /Iteration_1/Documents/Base de données/index.sql

Notamment :
Dans membre,

- On ajoute des contraintes *uniques* distinctes sur email et username car deux membres ne peuvent avoir en commun aucun des ces attributs.
- La connexion se fera avec l'username et le mot de passe et/ou l'adresse email et le mot de passe (à ce stade du développement de l'application ce point n'est pas encore décidé) on crée donc deux index, le premier sur mail puis password, le second sur username puis password.

Les monuments seront très souvent recherchés par rapport à leurs positions, on aura donc un index sur le n-uplet estTemporaire, lattitude, longitude.

Ajout de triggers

Un certains nombre de triggers peuvent être utiles pour rendre automatiques certaines modifications sur la base, lien du fichier les contenant :
/Iteration_1/Documents/Base de données/index.sql

trigger_contrib_moderateur : Vérifie que l'id membre dans l'attribut moderateurDemande correspond bien à un modérateur.