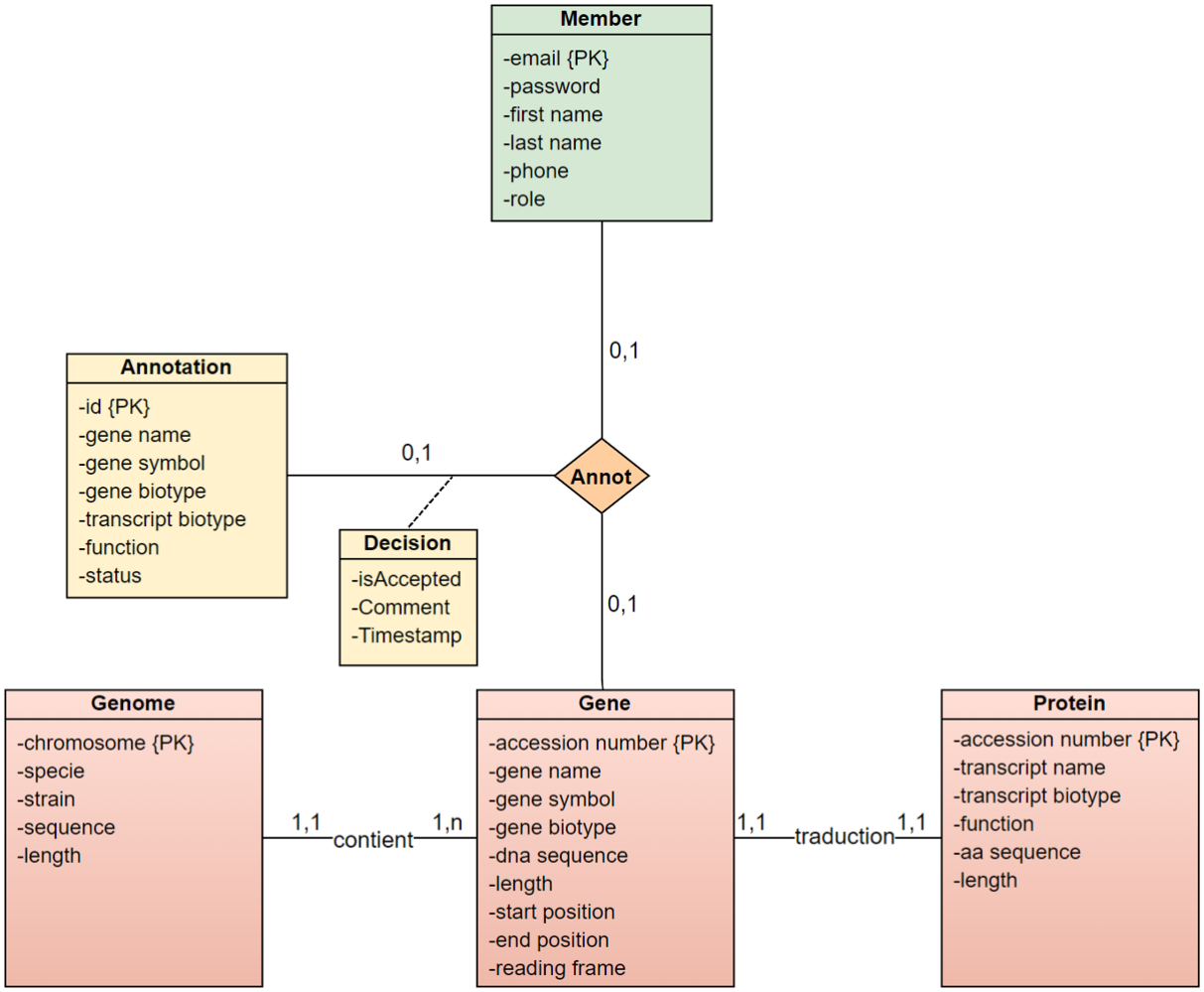
Base de données

L'application web d'annotation et d'analyse de génomes procaryotes requiert la conception et la création d'une base de données permettant de recenser l'ensemble des informations annotées par les membres. Cette base de données a été construite selon le cahier des charges du client et peut être modifiée si besoin. Celle-ci a pour objectif de répondre aux principales fonctionnalités attendues par le client et permettre d'effectuer des requêtes le plus rapidement possible. La première étape est la modélisation et la seconde est la conception.

Modélisation

La modélisation de la base de données commence par la création du diagramme UML illustré en Figure 1. Le diagramme UML est construit sur la base de trois sujets : le premier est la gestion des membres inscrits sur l'application (vert), le second concerne les informations relatives aux génomes bactériens (rouge) et le troisième concerne les annotations (jaune).

Figure 1 : diagramme UML d'annotation de génomes bactériens



La gestion des utilisateurs de l'application est traduite par la classe *member*. Nous disposons de plusieurs informations concernant les membres, à savoir leur email et un mot de passe sécurisé leur permettant de se connecter, ainsi que des informations moins importantes telles que leur prénom, nom et numéro de téléphone. Ces informations mineures permettent de personnaliser l'envoi de mails aux membres. Également, un membre a un rôle qui confère différents droits dans l'application :

lecteur, annotateur et validateur. Actuellement, rien ne justifie la création d'une classe pour chaque rôle qui hériterait de la classe *member*.

Deuxièmement, il convient de gérer les informations relatives aux génomes bactériens. Trois classes sont proposées à cet effet : *genome*, *gene* et *protein*. Ces trois classes sont très importantes puisque les membres de l'application pourront remplir un formulaire pour rechercher et obtenir des informations relatives aux séquences bactériennes qui les intéressent. La classe *genome* concerne l'ensemble des informations relatives au génome d'une bactérie. Nous retrouvons l'espèce, la souche, l'identifiant du chromosome, la séquence nucléotidique et la longueur du génome (unité : bp). L'identifiant du chromosome forme la clé primaire. Une alternative serait de proposer que la clé primaire soit l'espèce et la souche, mais l'identifiant du chromosome est unique pour chaque souche et un procaryote ne contient qu'un seul chromosome. Quant à la classe *gene*, celle-ci contient l'accession number qui est la clé primaire, ainsi que le nom du gène, le symbole génique, le biotype, la séquence nucléotidique, la longueur du gène (unité : bp), les positions de début et de fin dans le chromosome ainsi que le sens de lecture (direct ou inverse). Quant à la classe *protein*, celle-ci contient le nom du transcript, le biotype, la fonction de la protéine obtenue, la séquence en acides aminés ainsi que la longueur en acides aminés (unité : bp). Concernant les associations, un génome contient plusieurs gènes, mais un gène ne peut appartenir qu'à un seul génome. De plus, l'association gène et protéine est un-un puisqu'un gène est traduit en une protéine, et une protéine n'est traduite que par un gène.

La troisième partie non négligeable est la gestion de l'annotation des nouveaux génomes. Nous avons donc une classe *annotation* qui recense l'ensemble des annotations en cours, validées, ou rejetées (visible via l'attribut *status*) des séquences géniques et protéiques. Cette classe *annotation* dispose, pour chaque instance, d'un identifiant unique jouant le rôle de clé primaire ainsi que des informations annotables, à savoir le nom du gène, le symbole génique, le biotype génique, le biotype du transcrit et la fonction de la protéine obtenue après traduction. D'autres informations, à savoir la position de début et de fin d'un gène dans le génome, les longueurs des gènes et peptides, ou encore le sens de lecture sont déjà fournies par l'application qui automatise ces informations afin d'éviter toute erreur humaine et de disposer de ces informations avant même l'annotation.

Enfin, la classe *annot* joue un rôle central entre les trois grandes classes présentées précédemment (*member*, *genome*/*gene*/*protein* et *annotation*) puisqu'elle permet de créer une association ternaire entre ces différentes classes. Également, nous pouvons observer qu'un historique des décisions prises par les validateurs est prise en compte dans ce diagramme.

Conception

Maintenant que nous avons modélisé le cahier des charges à l'aide d'un diagramme UML, il convient d'en déduire le schéma relationnel lié à ce dernier, en veillant à ce que les tables permettent une optimisation des requêtes potentiellement les plus utilisées. Celui-ci est le suivant (en omettant les types de données qui seront décrits plus loin) :

member(<u>email</u>, password, first name, last name, phone, role)
genome(<u>chromosome</u>, specie, strain, sequence, length)
gene_protein(<u>accession_number</u>, dna length, start position, end position, reading frame, aa length, chromosome, isAnnotated)
gene_seq(<u>accession_number</u>, sequence)
protein_seq(<u>accession_number</u>, sequence)

annotation(accession_number, gene name, gene symbol, gene biotype, transcript biotype, function, status, email)
decision(accession_number, attempt_number, isApproved, comment, timestamp)

Une table *member* a été créée pour la gestion des utilisateurs de l'application web avec les mêmes attributs que ceux fournis dans le diagramme UML. La partie la plus compliquée à traduire du diagramme UML au schéma relationnel sont les classes relatives aux informations génomiques. La table *genome* a été créée avec les mêmes attributs que sa classe. En revanche, nous avons fusionné les classes gènes et protéines dans une unique table (*gene_protein*) puisque la relation est un-un. Nous pouvons remarquer, dans la table *gene_protein*, qu'il n'y a pas les attributs qui peuvent être annotés. Cela est dû au fait que nous ne souhaitons pas dupliquer les informations. Nous avons donc substitué l'ensemble de ces attributs par l'attribut isAnnotated qui est vrai si la séquence a été annotée et faux sinon. Une possibilité également, qui n'a pas été retenue, consistait à diviser les gènes et protéines annotées de ceux qui ne le sont pas (en créant deux tables distinctes) afin d'accélérer les recherches de séquences non annotées lorsque les validateur souhaite allouer des séquences à annoter aux annotateurs. Concernant la table *gene_protein* encore, nous avons séparé les séquences nucléotidiques et protéiques puisque ces attributs sont volumineux en termes de stockage et pourraient conduire, si aucun partitionnement n'était fait, au chargement de nombreuses pages même si les requêtes sont uniquement effectuées sur des champs qui ne concernent pas les motifs de séquence. Nous avons donc deux autres tables créées à cet effet : *gene_seq* et *protein_seq*. Concernant maintenant l'association entre le génome et ses gènes et protéines, nous avons passé la clé primaire chromosome de la table *genome* en clé étrangère dans la table *gene_protein* puisque nous avons une relation un-plusieurs. Concernant les annotations, nous avons une table *annotation* qui prend en clé primaire l'accession number plutôt qu'un identifiant auto-incrémenté. Ainsi, on peut faire directement une fusion entre les tables *annotation* et *gene_protein* sans passer par l'intermédiaire d'une autre table. La clé primaire d'accession number dans la table *anntation* est donc une clé étrangère de la clé primaire de la table *gene_protein*. Également, nous avons ajouté l'email de l'annotateur en tant que clé étrangère afin de faire directement le lien entre la table *Annotation* et *Member*. Une dernière table a été créé pour recenser l'historique des annotations : *decision*. Cette table contient l'accession number qui forme une clé primaire avec le numéro de tentative effectué par l'annotateur. Également, nous retrouvons le commentaire du validateur ainsi que l'horodatage de la décision.

Implémentation

L'implémentation de la base de données requiert la création de certains domaines dont nous les énumérons tous ci-dessous :

phone number (vérifie que le numéro est dans le format E.164)
accession number (non nul, vérifie que l'identifiant d'accès du gène commence par trois lettres puis est suivi de cinq chiffres)
genome sequence(non nul, vérifie que la sequence est effectivement une séquence nucléotidique de génome. Admet des valeurs N pour les nucléotidiques dont on ne connaît pas la réelle valeur)
gene sequence (non nul, vérifie que la séquence est effectivement une séquence nucléotidique)
protein sequence (non nul, vérifie que la séquence est effectivement une séquence protéique)

Également, certaines contraintes sont vérifiées :

Check_role_available (vérifie que le rôle prend la valeur *reader*, *annotator* ou *validator*)
Check_status_available (vérifie que le statut prend la valeur *ongoing*, *approved* ou *rejected*)

Unique_chromosome (vérifie l'unicité du chromosome)

L'utilisation d'index peut-être très important pour accélérer les requêtes. Il est à noter que certains index sont créés automatiquement, à savoir les index sur les clés primaires et attributs uniques. Par conséquent, nous n'avons pas besoin de créer manuellement un index sur l'attribut *chromosome* dans la table *genome*. L'index est déjà créé : ainsi, si un utilisateur connaît le nom du chromosome qui l'intéresse et souhaite récupérer son génome, la requête s'effectuera très rapidement. Nous n'avons donc créé qu'un seul index : Index_strain qui est une table de hachage sur l'attribut *strain* dans génome. Ainsi, il est possible que l'utilisateur renseigne uniquement la souche, sans prendre la peine de spécifier l'espèce dont il s'agit. En pratique, une souche a un nom unique (ou presque). Avoir un index sur la souche permet donc d'accélérer les requêtes où l'individu renseigne la souche qui l'intéresse alors qu'il n'a pas renseigné l'espèce.