Spécifications fonctionnelles et techniques

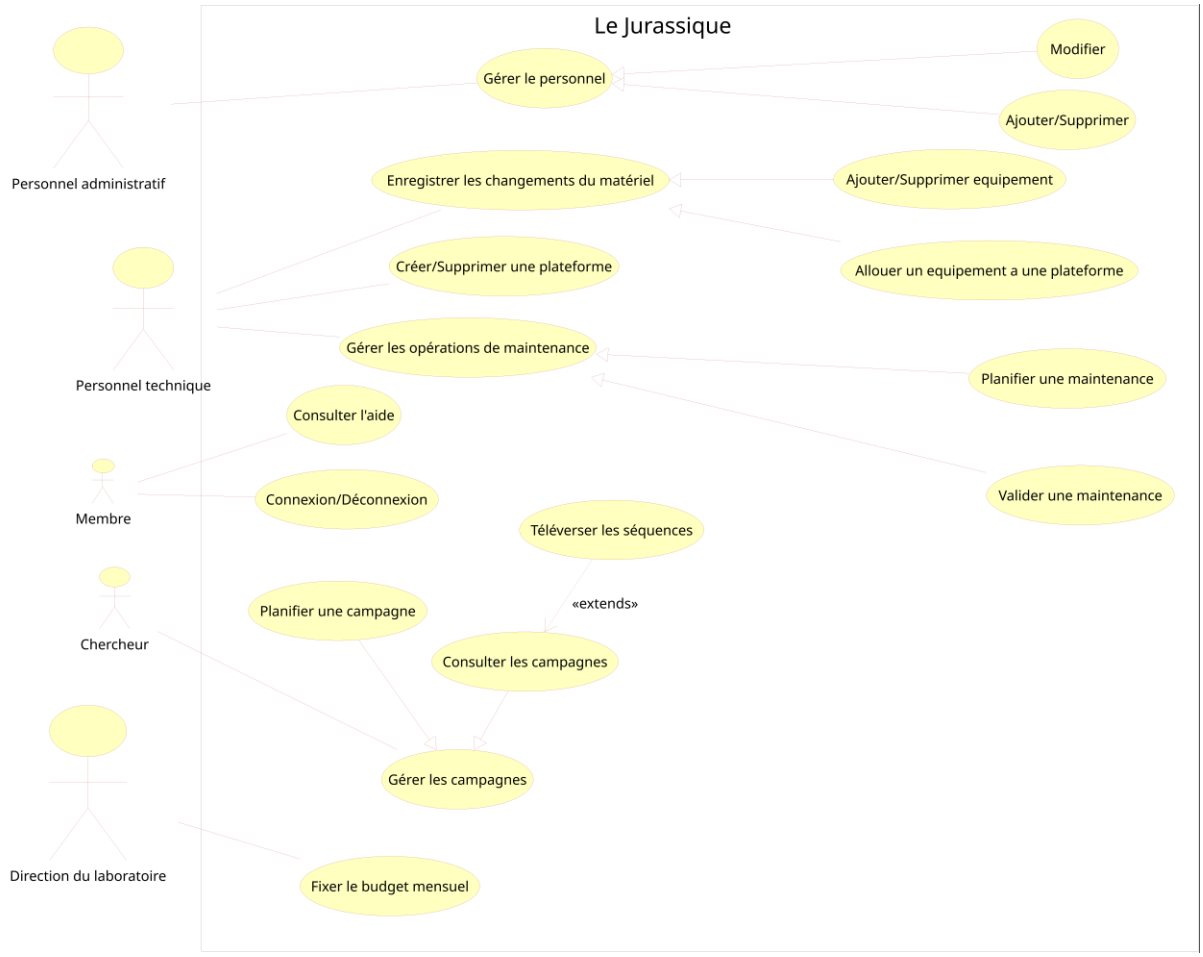
Spécifications fonctionnelles :

Description générale du système :

Le système de l'application consiste à un site web, une base de données. Le but de cette application est de fournir à un laboratoire de paléontologie une application qui leur permet de planifier des campagnes de fouilles et de les gérer efficacement, mais également exploiter les résultats de ces fouilles. Ce système contient de nombreux acteurs qui utiliseront cette application. En effet, les différents acteurs seront les personnels administratifs et techniques qui s'occuperont respectivement de gérer le personnel et l'équipement. De plus, les personnels techniques s'occuperont de créer ou supprimer des plateformes ainsi que gérer les opérations de maintenance. Les chercheurs pourront planifier sans conflit des campagnes de fouilles et téléverser des fichiers de séquences afin que ces derniers soient traités par le système et enfin la direction du laboratoire s'occupera de fixer le budget mensuellement ce qui va limiter la quantité de campagne créés

Exigences fonctionnelles :

Pour représenter ce que notre application devait être capable de réaliser, nous avons créé un diagramme des cas d'utilisations de notre application. Nous avons donc représenté toutes les actions que le personnel sera amené à faire. Ci-dessous le digramme de cas d'utilisation de notre application.



Ainsi, dans ce digramme CU on a représenté les rôles du personnel par des acteurs et également un acteur nommé ”Membre” qui représente un utilisateur non authentifié ou un simple visiteur, chaque acteur à des actions qu’il peut réaliser selon son rôle. Nous avons également utilisé la généralisation pour représenter des actions qui font partie d’un parent. Ainsi, grâce à ce diagramme on peut s’y référer pour rapidement voir quelles sont les actions que les utilisateurs seront capables de faire faire quoi une fois l’application entièrement fonctionnelle.

Une fois le diagramme des cas d’utilisations réalisés, nous avons produit des scénarios et un diagramme d’activité pour représenter les différentes actions nécessaires à un cas d’utilisation du diagramme de cas d’utilisation produit ci-dessus.

Un scénario est représenté sous forme de trois types de scénario : nominal, alternatif et d’exceptions qui chacun est représenté par un tableau de deux colonnes, la première représente les actions de l’utilisateur et le second représente les actions du système de l’application.

Les scénarios nominaux sont les scénarios principaux, celui qui effectuent la tâche la plus courte possible, les scénarios alternatifs représentent les scénarios qui effectuent la tâche avec des actions alternatifs, les scénarios d'exceptions sont les scénarios qui résulterait à un échec de la tâche.

On a décidé de faire un diagramme d'activités sur un scénario qui nous paraissait essentiel à visualiser sous forme de diagramme d'activité. Nous avons produit ces différents scénarios et diagrammes afin de pouvoir mieux visualiser les cas d'utilisations de notre application, celle-ci apporte une aide pour développer les actions du système et les interactions avec les utilisateurs.

Voici un exemple sur les scénarios produits :

- Planifier une campagne :

Scénario Nominal :

Acteur	Système
1.Clique sur "Planifier une campagne"	
	2.Redirige vers la page de planification d'une campagne
3.Choisit une date de début et de fin. Ajout du lieu, des personnes participantes, de la plateforme utilisée	
4.Clique sur "Planifier la campagne"	
	5.Le système affiche "Voulez-vous planifier cette campagne ?" (Oui/non)
6.Clique sur Oui	
	7. Le système affiche un message de confirmation et rediriger vers la page des campagnes

Scénario d'alternatif :

Acteur	Système
… 6. Clique sur Non	
	7. Le système annule l'action et reviens à la page de planification *Retour au point 4 du scénario nominal*

Scénarios alternatifs :

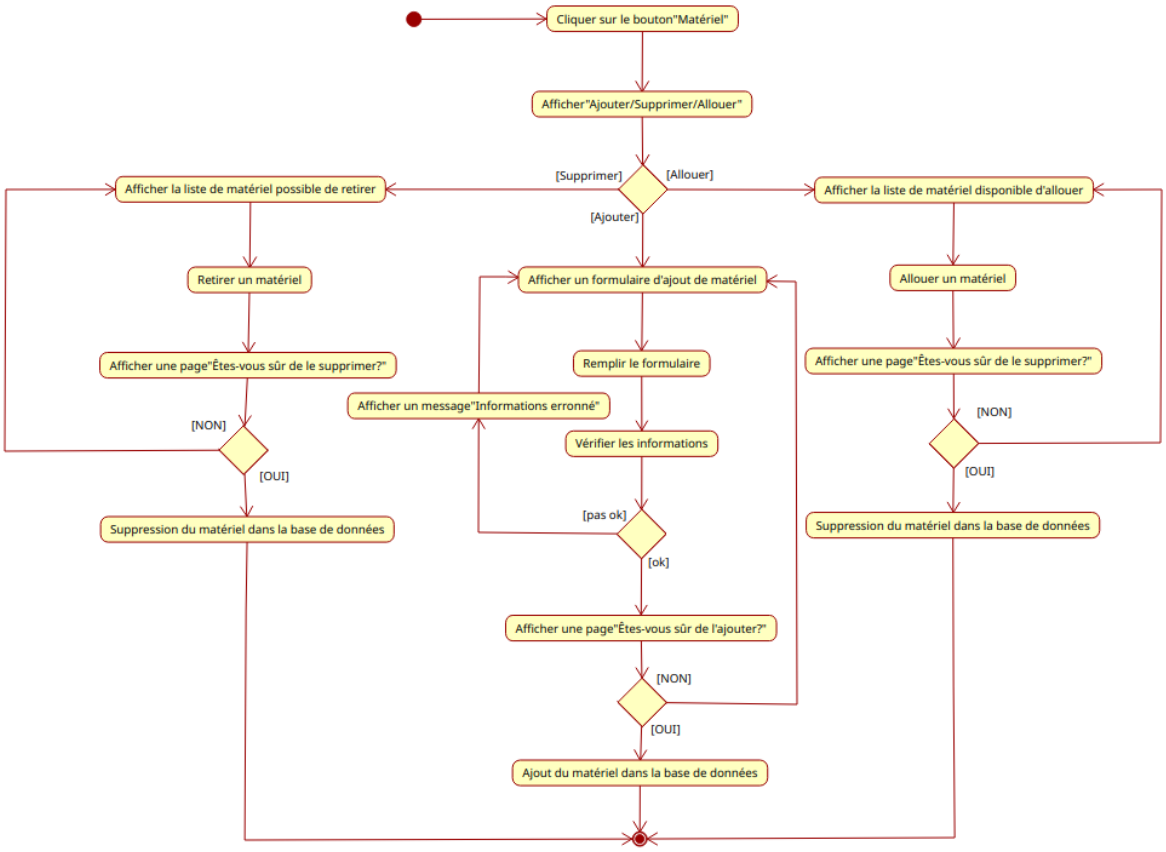
Acteur	Système
…	

3. Ajout du lieu, des personnes participantes, de la plateforme utilisée 4.Clique sur “Confirmer la campagne” 6.L’acteur met une date	 5. Le système met l’encart de date en rouge et affiche “veuillez préciser une date pour la campagne” *Retour au point 4 du scénario nominal*

Scénarios alternatifs :

Acteur	Système
3. Le directeur modifie le mois du budget à fixer 4. L’acteur inclus une somme qui dépasse le budget disponible du laboratoire 5. L’acteur clique sur le bouton “Confirmer” 7. L’acteur clique sur le bouton Oui 9. L’acteur clique sur le bouton Oui 11. L’acteur inclus une somme qui respecte le budget actuel du laboratoire 12. L’acteur clique sur le bouton “Confirmer” 14.L’acteur clique sur le bouton Oui	 6. Le Système renvoie un message de confirmation “Êtes-vous sûr de vouloir fixer ce budget pour le mois de [...] ?” (Oui/Non) 8. Le Système renvoie un message d’erreur “Le montant indiqué dépasse le budget disponible, Voulez-vous le modifier ? (Oui/Non) 10.Le Système retire le message d’erreur 13. Le Système renvoie un message de confirmation “Êtes-vous sûr de vouloir fixer ce budget pour le mois de [...] ?” (Oui/Non) 15. Le Système analyse les données du budget 16. Le Système renvoie l’acteur sur la page d’accueil avec une mini-page d’information “Le budget a été fixé avec succès !” et un bouton “Ok” *Fin du scénario*

Voici la représentation du diagramme d'activité du scénario "Enregistrer les modifications matériels" :



Interfaces et interactions :

Pour la première partie de cette SAE nous avons dû créer une maquette. Nous sommes d'abord partis sur une maquette réalisée sur Canva, cependant nous nous sommes rapidement rendu compte que Canva n'était pas parfaitement adapté au maquettage de site, par exemple Canva ne permet pas de modifier précisément chaque élément avec de nombreuses propriétés.

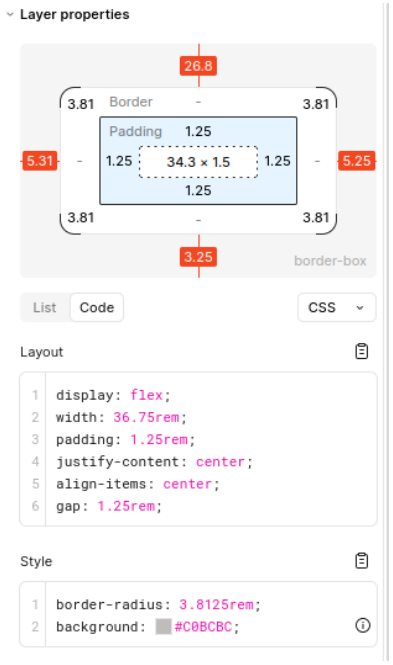
Nous avons donc décidé d'utiliser Figma, un outil de maquettage de site web qui est adapté à la création de site web avec des fonctions plus adaptés à ce que nous recherchions. Cet outil permet de palier aux manques de fonctionnalités de Canva.

Figma permet par exemple d'avoir une arborescence beaucoup plus détaillée et facile à gérer. On peut également gérer le padding des éléments. Cet outil, nous permet également d'avoir beaucoup plus de choix sur le style de notre site web et donc cela nous permet d'avoir beaucoup plus d'options de personnalisation.

Ci-dessous la page de connexion de la maquette :

Un des autres avantages et le fait de pouvoir avoir les propriétés CSS de chaque élément que l'on a créé de la même manière que lorsque que l'on souhaite inspecter un élément sur une page dans un navigateur.

Sur la capture d'écran ci-contre on peut voir le CSS ainsi que les proportions associé bouton "Se Connecter" de la page de connexion.

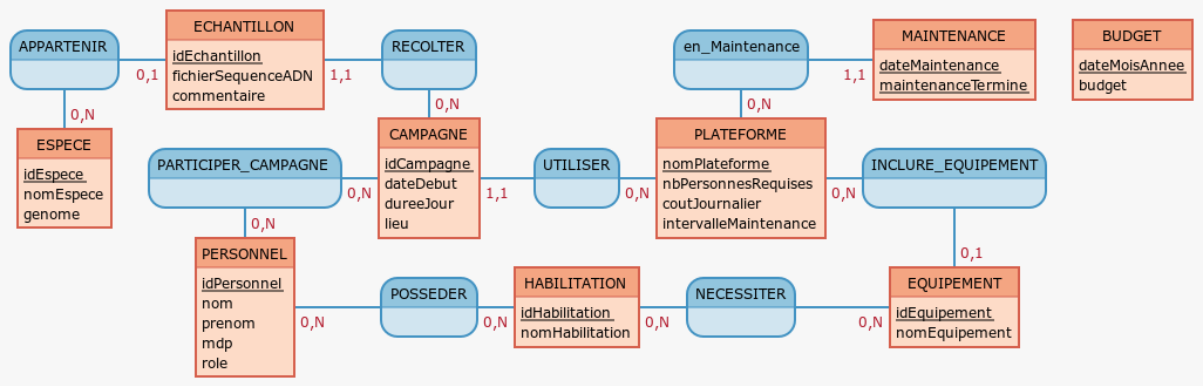


L'image ci-dessous correspond à la page du chercheur dans lequel il peut visualiser toutes les campagnes, filtrer selon le lieu, la plateforme utilisée, ou par ordre alphabétique. De plus, lorsque le chercheur survolera une ligne d'une campagne, cette ligne sera mise en gras et soulignée pour lui indiquer que s'il clique alors il pourra accéder à plus d'informations comme la liste complète des participants.

Par ailleurs, cette page est importante étant donné que le principe de ce tableau est réutilisé pour d'autres utilisateur comme le fait de devoir gérer le personnel mais adapté pour l'usage nécessaire associé, en modifiant par exemple les données affichées.

Modélisation de la base de données :

MCD (Modèle Conceptuel des Données) :



Explications du MCD :

Nous avons décidé de représenter les membres du personnel indépendamment de leurs position/poste dans une table *PERSONNEL* contenant un champ énuméré des différents postes possibles ainsi que les informations utiles sur une personne.

Ces personnes possèdent ou non, une ou plusieurs habilitations représentées par une table *HABILITATION*, le lien de possession d'une habilitation par un membre du personnel se fait par l'intermédiaire d'une association *POSSEDER*.

Il en est de même pour les équipements utilisés lors de fouilles, représenté par une table *EQUIPEMENT*, cet équipement peut nécessiter ou non d'une ou plusieurs habilitations pour être utilisé, cela est représenté sur notre MCD par l'association *NECESSITER*.

Afin d'organiser une campagne, nous avons modélisé les plateformes utilisées au cours ce celles-ci par la table *PLATEFORME*, cette plateforme contient des équipements, seulement un équipement ne peut être contenu que par une seule plateforme à la fois, nous avons représenté cela par une association *INCLURE_EQUIPEMENT*.

Toutes ces informations nous permettent de modéliser une campagne de fouille sous la forme de la table *CAMPAGNE*, une unique plateforme est attribuée à cette campagne et plusieurs membres du personnel peuvent y participer. Ces contraintes sont formalisées sous la forme de 2 associations : *UTILISER* et *PARTICIPER_CAMPAGNE*.

Cependant, une plateforme nécessite des maintenances périodiques, ces maintenances sont représentées par une table *MAINTENANCE*, et ne concernent qu'une seule plateforme à la fois, a contrario une plateforme subira de multiples maintenances au cours de son exploitation.

De plus, les échantillons récoltés au cours des campagnes une fois analysées doivent être sauvegardé en référençant la campagne d'où l'échantillon provient, pour cela nous avons décidé d'ajouter une table *ECHANTILLON*, qui n'est lié qu'a une unique campagne par l'association RECOLTER.

Nous avons aussi décidé qu'un échantillon peut être identifié et donc rattaché à une espèce représentée par la table *ESPECE*.

Enfin, les possibilités d'organisations de campagnes se calquent sur un budget alloué au mois, nous avons donc ajouté une table BUDGET.

Modification futur possible :

Par la suite, nous pensons ajouter la possibilité de faire le rapprochement entre plusieurs espèces et/ou échantillons, à la suite des traitements effectué par les algorithmes qui se verront implémenté.

Nous aimerions aussi pouvoir améliorer la modélisation des espèces au sein de la base de données.

Dépendances fonctionnelles :

La base de données possède plusieurs dépendances fonctionnelles, même si elle se concentre principalement sur les dépendances non-fonctionnelles, il est important de s'assurer que :

- Lors d'une campagne, elle doit forcément être appliqué à une plateforme
- Une plateforme ne peut pas inclure plusieurs équipements.

- Un échantillon doit être récolté par une campagne uniquement.

-Un échantillon appartient au maximum à une espèce

Les dépendances fonctionnelles s'écriront ainsi :

dateMoisAnnees -> budget

idEspece -> nomEspece, genome

idEchantillon -> fichierSequenceADN, commentaire, idCampagne, idEspece

nomPlateforme -> nbPersonnesRequises, coutJournalier, intervalleMaintenance

idCampagne ->dateDebut, dureeJour, lieu, nomPlateforme

idPersonnel -> nom, prenom, mdp, role

idEquipement -> nomEquipement

idHabilitation -> nomHabilitation

Dépendances non-fonctionnelles :

L'ensemble des fonctions, procédures et triggers produit sont présent dans le fichier "triggers.sql"

- Budget

Le budget doit suivre plusieurs contraintes vérifiées par des triggers :

Avant insertion (*INSERT*) :

- Il ne peut y avoir qu'un seul budget par mois.
- Le budget doit être une valeur positive.

Lors de l'insertion d'un budget, tout d'abord il ne peut être négatif autrement il s'agit d'une dette, une simple comparaison (budget > 0) vérifie cette contrainte.

De plus, il ne peut y avoir de multiple budget pour un même mois, pour cela on vérifie qu'une requête sélectionnant les budgets à la date (mois et année) du budget inséré retourne un ensemble vide ce qui signifie qu'il n'y a pas de budget pour ledit mois.

Avant modification (*UPDATE*) :

- Un budget ne peut être modifié si le nouveau budget ne permet pas de couvrir les fonds investis dans les campagnes courante.
- Le budget doit être une valeur positive.
- Un budget ne peut être modifié une fois le mois de celui-ci écoulé.

Si un budget vient à être modifié, il doit couvrir les fonds déjà engagés, soit la nouvelle valeur du budget doit être supérieur ou égale à la somme des fonds engagé obtenu grâce à une fonction.

De même, le budget modifié doit être le budget du mois courant, autrement cela ne ferait pas sens, nous vérifions alors que le mois et l'année de la date actuelle n'est pas inférieur à la date du budget.

Avant suppression (*DELETE*) :

- Un budget ne peut être supprimer si des fonds sont engagé.

En effet, une part du budget est déjà alloué, ce budget n'est plus supprimable, seulement modifiable, on vérifie simplement qu'il n'y a pas de campagnes commençant au cours du mois concerné.

- Campagne

Lors de la création d'une campagne, il faut s'assurer que :

- Le coût d'une campagne ne doit pas excéder le budget restant du mois du démarrage de celle-ci.
- Une plateforme n'est pas en cours de maintenance lors du démarrage d'une campagne.
- La plateforme utilisée durant une campagne ne doit pas avoir besoin de maintenance au cours de celle-ci (la durée de la campagne ne fait pas dépasser l'intervalle de maintenance).
- Lors d'une campagne, les personnes et la plateforme ne doivent pas être occupé par une autre campagne simultanément.

Ces contraintes sont assurées par un trigger qui filtre les insertions.

Tout d'abord, pour pouvoir préparer une campagne le coût de celle-ci doit être supérieur ou égal au budget restant du mois calculé au travers d'une fonction.

En supplément, il faut vérifier que la plateforme choisis pour la campagne ne nécessitera pas de maintenance au cours de la campagne et qu'elle ne soit pas en cours de maintenance au début de la campagne.

- Equipement

Un équipement doit vérifier les contraintes suivantes :

- Un équipement ne peut être utilisé par plusieurs plateformes simultanément.

Cette contrainte est vérifiée par 2 triggers, a l'insertion (*INSERT*) et à la modification (*UPDATE*)

- ### Participation des chercheurs

Nous avons créé un trigger vérifiant que :

- Un chercheur ne peut participer qu'a une seule campagne en simultané

Il suffit donc de s'assurer que toutes les participations de ce chercheur ne se superpose pas à la nouvelle campagne.

- ### Validation des campagnes

Une campagne peut se voir validé ou invalidé qu'après inscription ou retrait de participation ladite campagne car il faut vérifier :

- Le nombre de participant respectant le nombre minimal de personnes nécessaire à l'utilisation de la plateforme associée à la campagne
- Toutes les habilitations liées aux équipements présents sur la plateforme associée sont représentées au travers des participants

Pour cela on vérifie grâce à un trigger post-insertion (*AFTER INSERT*) et post-suppression (*AFTER DELETE*) qui modifie la valeur du champs valide de la campagne si elle respecte les conditions énoncées et inversement.

Fonctions et Procédures :

Afin de simplifier l'implémentation de certains triggers cité précédemment nous avons établis une suite de fonctions :

- plateformeInMaintenance()

Cette fonction permet de déterminer si une plateforme est en cours de maintenance pour une date donnée, ou nécessitera une maintenance au cours d'une campagne.

Elle prend en paramètres la plateforme, date d'utilisation, ainsi que la durée d'utilisation de celle-ci, et retourne un booléen selon si elle peut être utilisé ou non.

- plateformeAvailable()

La fonction plateformeAvailable() nous indique si la plateforme n'est pas déjà en cours d'utilisation sur une autre campagne de fouille.

Elle prend en paramètre la date d'utilisation de la plateforme ainsi que son nom, retournant un booléen selon qu'elle soit disponible ou non pour la campagne.

- remainingBudget()

Elle prend en paramètre une date, et permet de calculer le budget restant alloué pour le mois de la date donnée en argument.

Elle retourne une valeur décimale.

- verifyHabilitationValidity()

Enfin, cette fonction permet de déterminer si une habilitation donnée est représenté au sein des participants d'une campagne.

Elle prend en paramètre le numéro de ladite campagne, ainsi que l'habilitation à vérifier, elle retourne un booléen selon que l'habilitation soit représentée ou non.

Indexe

Au premier abord, nous avions décidé de supprimer les campagnes invalide au jours de leur commencement.

Pour cela nous avions décidé d'utiliser un "*EVENT*" qui aurait permis de supprimer lesdites campagne ainsi que les participations à cette campagne grâce à la procédure "*clearCampaign()*".

```
create or replace EVENT deleteInvalidCampaign ON SCHEDULE EVERY 1 MINUTE DO
    call clearCampaign();
```

Cependant nous avons abandonné cette idée pour 2 raisons majeurs, la première étant que la mise en place d'un event utilisant un Scheduler nécessite d'activer un paramètre de niveau "super-privilège", ce qui n'est pas possible en utilisant le serveur de base de données utilisé.

La seconde raison, est qu'après discussions et une longue réflexion nous avons décidé de garder en mémoire toutes les CAMPAGNE.

Nous avons ajouté en conséquence un index sur le champ "valide" de la table *CAMPAGNE*, nous permettant d'accéder facilement et rapidement aux campagnes intéressantes pour certains cas d'utilisation à moindre coût.

Spécifications techniques :

Architecture :

Pour le développement de notre application web, nous avons mis en place une architecture logicielle basée sur le langage Python et le Framework Flask.

Nous avons choisi Flask car c'est un "micro-Framework" léger. Il nous a permis de structurer l'application selon nos besoins spécifiques sans la lourdeur d'outils plus complexes. De plus, notre maîtrise préalable de cet outil a facilité la prise en main et accéléré le développement.

Pour la gestion de la base de données, l'infrastructure repose sur MariaDB. Pour faire le lien entre notre code Python et la base de données, nous utilisons SQLAlchemy. Cet ORM est primordial : il nous permet d'utiliser les tables de la base de données comme des objets Python.

Enfin, l'interface utilisateur est conçue en HTML et CSS pour l'affichage des pages, qui sont générées dynamiquement par le serveur Flask.

L'application fonctionne sur une architecture classique Client-Serveur. Dans le cadre de ce projet, le serveur web et le serveur de base de données sont hébergés sur la même machine. Ils communiquent en local, tandis que l'utilisateur accède à l'application via son navigateur.

Gestion des interfaces :

Le protocole de communication principal que nous utilisons est le protocole HTTP en envoyant des requêtes au serveur Flask, qui y répond. Nous utilisons 2 types de méthodes HTTP sur notre application, la première étant le “GET”, utilisé pour demander et afficher des pages ; et le “POST” que l’on utilise afin d’envoyé des données serveurs comme ajouter un personnel à partir du formulaire.

Notre vue implémente principalement l’opération CRUD ou nous faisons de l’ajout (CREATE :add_personnel, create_campaign,...), de la lecture de données (READ : gestion_personnel, platform_management,...), de la modification d’informations (UPDATE : edit_personnel, edit_campaign,...) et de la suppression de donnée(s) (DELETE : delete_plateforme, delete_sample,...)

À l’aide de Flask-WTF, on peut permettre l’échange de données du site au serveur, lorsque l’on soumet un formulaire, la bibliothèque Flask-WTF va décoder automatiquement pour le rendre accessible via “request.form”. Quant au serveur, le serveur envoie dans la grande majorité des documents HTML comme le remplissage d’une table avec des données de notre base de données (une partie minime envoie du json pour des erreurs de création).

Nous utilisons la bibliothèque Flask-login afin de créer une session quand un utilisateur se connecte. Cette session est maintenue grâce à un cookie envoyé au navigateur. Ce cookie contient un identifiant de session signé numériquement avec un SECRET_KEY.

Pour finir, nous avons implémenter un contrôleur d’accès via @login_required pour nous assurer à ce que l’utilisateur doit être connecté afin d’accéder au site, mais nous

avons aussi ajouter des @role_access_rights(ROLE.[...]) afin de spécifier que la page ne peut être accessible qu'à partir du rôle spécifié.

Normes et standards :

Afin de rendre le code plus lisible et de trouver des moyens plus efficaces d'implémenter notre code, nous avons utilisé plusieurs outils permettant d'améliorer la qualité du code tel que Pylint et SonarLint, ainsi que des conventions de développement comme l'arborescence de notre programme et l'utilisation de GitHub.

Nous avons utilisé un outil permettant de compléter l'outil suivant automatiquement, l'outil yapf permettant de modifier certaines erreurs qui ne suivent pas le PEP 8, qui laisse 4 espaces avant chaque commentaire et bien d'autres petites modifications pour rendre le code plus lisible.

Nous nous sommes servies de PyLint puisque nous utilisions comme langage de programmation Python pour permettre un code python de qualité, suivant les conventions PEP 8 (snake_case, empty space,...)

L'utilisation de SonarLint a aussi complémenter PyLint et permet de pouvoir rendre une qualité de code optimale durant le développement (car c'est une extension Visual Studio Code).

La documentation de notre programme est directement écrite sur le readme.md du GitHub permettant ainsi d'expliquer à l'utilisateur comment faire fonction l'application web et expliquer ces aspects techniques.

Nous avons créé des issues sur GitHub afin de répartir les tâches mais aussi de savoir les tâches qui restait à faire, ce qui nous a permis d'être plus performant et ne pas se perdre dans le développement. Elles nous ont aussi permis de créer des branches pour chaque tâche.

Environnement :

Pour réaliser cette SAE nous avons utilisé à l'iut le serveur de base de données MariaDB et chez nous, nous utilisons Docker avec un container contenant une base de données de Test et un autre container contenant une base de données destiné à l'application. Qui sont créé par un fichier ".yml" qui permet d'avoir les mêmes containers et donc d'avoir une base uniforme. Ainsi grâce à cela nous avons juste à modifier le fichier de configuration pour accéder à la base de données et tout le reste fonctionne.

Nous utilisons également Git avec GitHub afin de sauvegarder notre projet et d'avoir chacun notre espace de développement distinct en suivant la méthode Git flow.

De plus, nous avons réalisé des releases pour la partie algorithmique et pour l'application finale.

En ce qui concerne les tests nous avons réalisés les tests de l'ORM, des pages html et des formulaires implémentés à l'aide de Pytest et de la couverture de Coverage pour nous assurer que nos implémentations produisent l'effet escompté.

Exigences de validation :

Afin de considérer cette application valide, nous avons définis que les méthodes et class développées devaient être à hauteur de 80%.

Ce qui signifie que le taux de coverage doit être de 80% au minimum.

Ce que nous avons réussi :

File ▲	statements	missing	excluded	coverage
config.py	2	0	0	100%
LaboDino\app.py	9	0	0	100%
LaboDino\decorators.py	17	3	0	82%
LaboDino\forms.py	136	43	0	68%
LaboDino\models.py	166	12	0	93%
LaboDino\views.py	293	65	0	78%
Total	**623**	**123**	**0**	**80%**

coverage.py v7.11.3, created at 2025-11-30 15:12 +0100

Améliorations possibles :

De multiples améliorations peuvent être apporté à l'application.

Notamment l'implémentation des traitements des échantillons d'ADN, et de l'importation de ceux-ci au sein de l'application.

Une amélioration du formatage des messages d'erreurs SQL afin de pouvoir mieux les présenter à l'utilisateur.

Ainsi qu'une amélioration globale du style et du dynamisme des pages web.

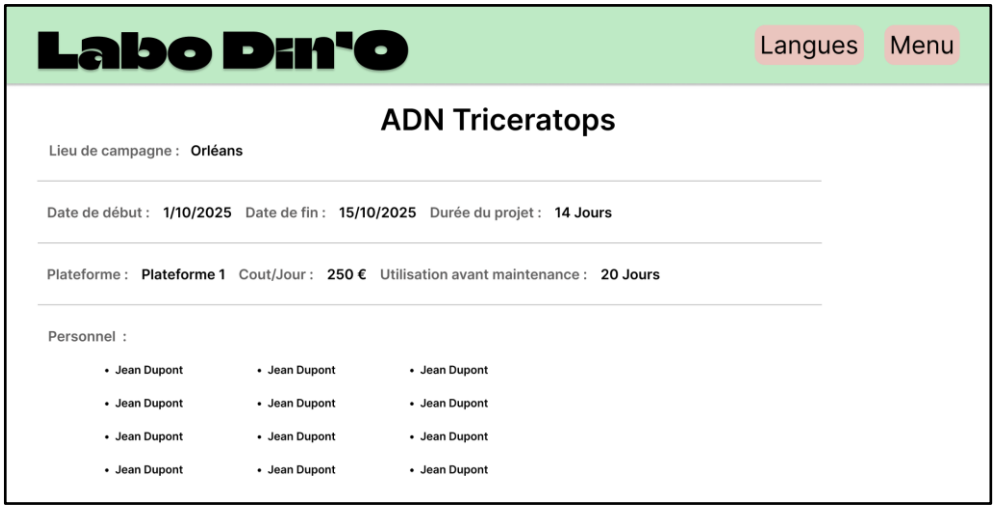
D'autres fonctionnalités pourront être implémentées, tels que la possibilité de spécifier de multiples habilitations nécessaires pour un même équipement, permettre à l'utilisateur de connecter rapidement, notamment avec les cookies, ainsi que l'implémentation d'une "search bar" dans le but de simplifier la recherche d'éléments.

Annexe :

Maquette du site web sur Figma :

Page de la gestion des plateformes du personnel technique :

Page détaillée d'une campagne lors du clic sur une des campagnes de la page de gestion globale du chercheur :



Page de la gestion du personnel du personnel administratif :

Liens de la vidéo de présentation:

Vidéo de présentation