Introduction

Ce rapport individuel présente l'analyse de mes contributions au projet de SAÉ visant à développer une application web de gestion du matériel d'un laboratoire de paléontologie. L'objectif principal de l'outil est de rendre plus efficace pour gérer les fouilles et les échantillons.
Mes réalisations se sont concentrées sur l'implémentation des espèces hypothétiques et avérées (classes, calcul de distance), la reconstruction d'un arbre phylogénetique et le refactoring de tout les algorithmes python

Compétences et Apprentissages Critiques Mobilisés

Les tableaux suivant détaille la manière dont mes actions ont satisfait les différents Apprentissages Critiques (AC) évalués lors de cette SAÉ. Vous pouvez retrouver mes conceptions pour la contribution au projet juste après les tableaux

1. RÉALISER (Conception et Implémentation)

AC	Contribution Réalisée	Justification
AC21.01	**Implémentation de la logique de filtrage du personnel (par nom, prénom, rôle) et gestion des droits d'accès.**	Traduction des exigences de sécurité et de gestion en implémentant des filtres dynamiques et des restrictions d'accès via les décorateurs @login_required et @role_access_rights(ROLE.administratif).
AC21.03	**Sécurisation des routes Flask et mise en place d'une suite de tests complète (Unitaires et d'Intégration)**	Adoption de bonnes pratiques : 1. **Sécurité :** Utilisation de décorateurs pour protéger les routes. 2. **Qualité :** Développement de tests unitaires (test_personnel, test_posseder) et vérification du code coverage (coverage.py) pour assurer la robustesse de l'application.

2. OPTIMISER

AC	Contribution Réalisée	Justification
AC22.01	**Modélisation complexe de la base de données avec SQLAlchemy (Tables d'association et Enum).**	Choix de structures adaptées pour gérer les relations Many-to-Many (ex: PARTICIPER_CAMPAGNE liant CAMPAGNE et PERSONNEL, POSSEDER pour les habilitations) et utilisation de types stricts comme Enum(ROLE) pour garantir l'intégrité des données.

3. ADMINISTRER

AC	Contribution Réalisée	Justification
AC23.01	**Développement des routes Flask communiquant avec la base de données et les templates.**	Conception d'une application communicante où le contrôleur (Flask) intercepte les requêtes HTTP (GET/POST), interroge la BDD via l'ORM, et renvoie les données filtrées au client.

4. GÉRER (Traitement des Données)

AC	Contribution Réalisée	Justification
AC24.03	**Création du template de gestion du personnel et validation de l'affichage via BeautifulSoup.**	Organisation de la restitution : 1. Visuel : Intégration de la pagination et des tableaux HTML. 2. Validation : Utilisation de BeautifulSoup dans les tests (test_gestion_personnel_get_with_filter) pour parser le HTML généré et vérifier programme la présence exacte des données attendues (ex: vérifier que 'Belobog' est bien dans la première cellule)
AC24.04	**Manipulation de données hétérogènes dans l'ORM et**	Gestion simultanée de types de données variés : chaînes de caractères (noms), dates

AC	Contribution Réalisée	Justification
	les vues.	(dateDebut), booléens (valide), et énumérations (ROLE), assurant leur cohérence entre le Python et la base SQL.

5. CONDUIRE (Gestion de Projet)

AC	Contribution Réalisée	Justification
AC25.02	**Rédaction du rapport commun (explication des fonctions et classes).**	La participation au rapport a permis de formaliser les besoins techniques et les solutions implémentées (mes classes et algorithmes) pour le reste de l'équipe et les évaluateurs.

6. COLLABORER (Travail d'Équipe)

AC	Contribution Réalisée	Justification
AC26.02	**Utilisation d'Outil de Gestion de projet : GitHub, Trello, Instagram**	Trello : Lister les tâches à faire, partager les ressources pour l'envoie des rapports perso, spécifications fonctionnelles et techniques GitHub : Travailler en équipe sur un projet dans la programmation de l'application, attribuer des tâches technique à chacun Instagram : Communiquer, Échanger des information entre les membres de l'équipe
AC26.03	**Rédaction du rapport des spécifications techniques et fonctionnelles.** **Rédaction du guide de l'utilisateur** **Co-développement de l'ORM et intégration de modules tiers (Pagination).**	Mobilisation des compétences interpersonnelles pour échanger, expliquer mon travail et le consolider dans un document commun et démontrer sa capacité à expliquer les fonctionnalité disponible sur l'application web Collaboration technique pour la conception du modèle de données (ORM réalisé en binôme) et intégration du système de pagination développé par un collègue au sein de mon propre template "personnel administratif".
AC26.04	**Réalisation de ce rapport individuel**.	Ce document sert à rendre compte de mon activité professionnelle (mes tâches et les compétences

AC	Contribution Réalisée	Justification
		mobilisées) de manière structurée et synthétique.

Conceptions Produites ou en Collaborations :

Implémentation de l'ORM :

```
class PERSONNEL(db.Model, UserMixin):

    __tablename__ = "PERSONNEL"
    id_personnel = db.Column("idPersonnel", db.Integer, primary_key=True,
autoincrement=True)
    nom = db.Column(db.String(50))
    prenom = db.Column(db.String(50))
    mdp = db.Column(db.String(10), unique=True)
    role = db.Column(db.Enum(ROLE))

    participerCampagne = db.relationship("PARTICIPER_CAMPAGNE",
back_populates="personnel", cascade="all, delete-orphan")
    posseder = db.relationship("POSSEDER", back_populates="personnels", cascade="all,
delete-orphan")

    def __init__(self, nom, prenom, mdp, role):
        self.nom = nom
        self.prenom = prenom
        self.mdp = mdp
        self.role = role

    def get_id(self):
        return self.id_personnel

    def get_role(self) -> ROLE:
        return self.role

    def __repr__(self):
        return f"<{self.nom} {self.prenom} : {self.role}>"

class PARTICIPER_CAMPAGNE(db.Model):
    __tablename__ = "PARTICIPER_CAMPAGNE"
    id_campagne = db.Column("idCampagne", db.Integer,
db.ForeignKey("CAMPAGNE.idCampagne"),primary_key=True)
    id_personnel = db.Column("idPersonnel", db.Integer,
db.ForeignKey("PERSONNEL.idPersonnel"),primary_key=True)
    campagne = db.relationship("CAMPAGNE", back_populates="participerCampagne")
    personnel = db.relationship("PERSONNEL", back_populates="participerCampagne")

    def __init__(self, idCampagne, idPersonnel):
```

```
    self.idCampagne = idCampagne
    self.idPersonnel = idPersonnel

  def __repr__(self):
    nom = getattr(self.personnel, 'nom', None)
    prenom = getattr(self.personnel, 'prenom', None)
    return f"<POSSEDER campagne= n°{self.idCampagne} personnel={nom!r} {prenom!r}>"


class CAMPAGNE(db.Model):

  __tablename__ = 'CAMPAGNE'
  id_campagne = db.Column("idCampagne", db.Integer, primary_key=True,
autoincrement=True)
  nom_plateforme = db.Column("nomPlateforme", db.String(50),
db.ForeignKey("PLATEFORME.nomPlateforme"))
  dateDebut = db.Column(db.DATE)
  duree = db.Column(db.Integer)
  lieu = db.Column(db.String(100))
  valide = db.Column(db.Boolean)
  participerCampagne = db.relationship("PARTICIPER_CAMPAGNE",
back_populates="campagne")
  plateforme = db.relationship("PLATEFORME", back_populates="campagnes")
  echantillons = db.relationship("ECHANTILLON", back_populates="campagne")
```

Implémentation de l'ORM :

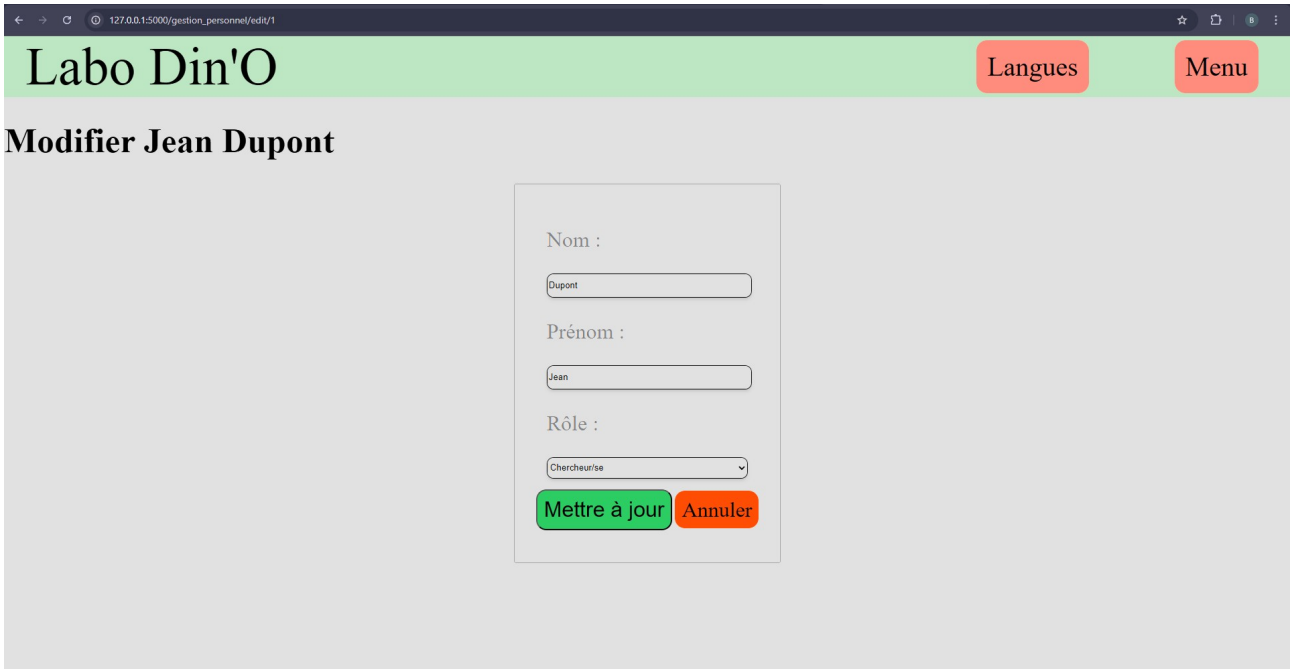
```
def test_personnel():

  personnel1 = PERSONNEL("Dupont", "Jean", "mdp123", ROLE.administratif)
  personnel2 = PERSONNEL("Martin", "Sophie", "mdp456", ROLE.chercheur)
  personnel3 = PERSONNEL("Bernard", "Luc", "mdp789", ROLE.technicien)
  personnel4 = PERSONNEL("Dubois", "Marie", "mdp101", ROLE.direction)

  db.session.add_all([personnel1, personnel2, personnel3, personnel4])
  db.session.commit()

  assert PERSONNEL.query.count() == 4
  assert PERSONNEL.query.filter_by(nom="Dupont").first().prenom == "Jean"
  assert PERSONNEL.query.filter_by(nom="Martin").first().role == ROLE.chercheur
  assert PERSONNEL.query.filter_by(role=ROLE.direction).count() == 1


def test_campagnes():

  plat1 = PLATEFORME("Plateforme Sequence", 5, 1500, 90)
  plat2 = PLATEFORME("Plateforme Paléontologie", 3, 800, 60)
  plat3 = PLATEFORME("Plateforme Analyse", 4, 1200, 45)

  db.session.add_all([plat1, plat2, plat3])
  db.session.commit()
```

```
    plateforme_sequence = PLATEFORME.query.filter_by(nom_plateforme="Plateforme Sequence").first()
    plateforme_paleontologie = PLATEFORME.query.filter_by(nom_plateforme="Plateforme Paléontologie").first()
    plateforme_analyse = PLATEFORME.query.filter_by(nom_plateforme="Plateforme Analyse").first()

    camp1 = CAMPAGNE(plateforme_sequence.nom_plateforme, date(2024, 1, 15), 30, "Montana, USA", True)
    camp2 = CAMPAGNE(plateforme_paleontologie.nom_plateforme, date(2024, 3, 10), 45, "Patagonie, Argentine", True)
    camp3 = CAMPAGNE(plateforme_analyse.nom_plateforme, date(2024, 6, 1), 20, "Gobi, Mongolie", False)

    db.session.add_all([camp1, camp2, camp3])
    db.session.commit()

    assert CAMPAGNE.query.count() == 3
    assert CAMPAGNE.query.filter_by(lieu="Montana, USA").first().valide == True
    assert CAMPAGNE.query.filter_by(lieu="Gobi, Mongolie").first().valide == False
    assert camp1.id_campagne is not None
    assert camp1.plateforme.nom_plateforme == "Plateforme Sequence"


def test_participer_campagne():

    """Teste l'association PARTICIPER_CAMPAGNE entre CAMPAGNE et PERSONNEL."""

    personnel1 = PERSONNEL("Dupont", "Jean", "mdp123", ROLE.administratif)
    personnel2 = PERSONNEL("Martin", "Sophie", "mdp456", ROLE.chercheur)
    personnel3 = PERSONNEL("Bernard", "Luc", "mdp789", ROLE.technicien)

    db.session.add_all([personnel1, personnel2, personnel3])
    db.session.commit()

    plat1 = PLATEFORME("Plateforme Sequence", 5, 1500, 90)
    plat2 = PLATEFORME("Plateforme Paléontologie", 3, 800, 60)

    db.session.add_all([plat1, plat2])
    db.session.commit()

    camp1 = CAMPAGNE(plat1.nom_plateforme, date(2024, 1, 15), 30, "Montana, USA", True)
    camp2 = CAMPAGNE(plat2.nom_plateforme, date(2024, 3, 10), 45, "Patagonie, Argentine", True)
    db.session.add_all([camp1, camp2])
    db.session.commit()

    personnel = PERSONNEL.query.all()
    campagnes = CAMPAGNE.query.all()

```

```
    part_camp1 = PARTICIPER_CAMPAGNE(campagnes[0].id_campagne,
personnel[0].id_personnel)
    part_camp2 = PARTICIPER_CAMPAGNE(campagnes[1].id_campagne,
personnel[1].id_personnel)
    part_camp3 = PARTICIPER_CAMPAGNE(campagnes[0].id_campagne,
personnel[2].id_personnel)
    part_camp4 = PARTICIPER_CAMPAGNE(campagnes[1].id_campagne,
personnel[2].id_personnel)

    db.session.add_all([part_camp1, part_camp2, part_camp3, part_camp4])
    db.session.commit()

    bernard = PERSONNEL.query.filter_by(nom="Bernard").first()

    assert PARTICIPER_CAMPAGNE.query.count() == 4
    assert
PARTICIPER_CAMPAGNE.query.filter_by(id_personnel=bernard.id_personnel).count() == 2
    assert len(bernard.participerCampagne) == 2
```

Template Gestion du personnel :

Template Modifier un personnel :



Implémentation du tableau HTML:

```
      <table id="table-admin">
        <thead>
          <th name="nom">Nom</th>
          <th name="prenom">Prénom</th>
          <th name="role">Rôle</th>
          <th>Action</th>
        </thead>
        <tbody>
          {% for personnel in personnels %}
            <tr>
              <td>{{ personnel.nom }}</td>
              <td>{{ personnel.prenom }}</td>
              <td>{{ personnel.role.value }}</td>
              <td>
                <form action="{{ url_for('erase_personnel',
id_personnel=personnel.id_personnel) }}" method="post" style="display: inline;">
                  <input type="hidden" name="id_personnel"
value="{{ personnel.id_personnel }}">
                  <button type="submit" class="action-btn" onclick="return
confirm('Êtes-vous sûr de vouloir supprimer cette personne ?');">
                    <i class="fa-solid fa-trash"></i>
                  </button>
                </form>
                <a href="{{ url_for('show_edit_form',
id_personnel=personnel.id_personnel) }}" class="action-btn">
                  <i class="fa-solid fa-pen-to-square"></i>
                </a>
              </td>
            </tr>
```

```
          {% endfor %}
        </tbody>
      </table>
```

implémentation de la views.py :

```
@app.route('/gestion_personnel/', methods=['GET', 'POST'])
@login_required
@role_access_rights(ROLE.administratif)
def gestion_personnel():
    if request.method == 'POST':
        filtre = request.form.get('filtre')
    else:
        filtre = request.args.get('filtre')

    # On commence par une requête de base
    query = PERSONNEL.query

    # On applique le tri en fonction du filtre
    if filtre == 'nom':
        query = query.order_by(PERSONNEL.nom)
    elif filtre == 'prenom':
        query = query.order_by(PERSONNEL.prenom)
    elif filtre == 'role':
        query = query.order_by(PERSONNEL.role)
    else:
        # Tri par défaut si aucun filtre n'est sélectionné
        query = query.order_by(PERSONNEL.id_personnel)

    # On exécute la requête finale
    personnels = query.all()

    page = request.args.get('page', 1, type=int)
    personnels_paginated, page = _pagination(personnels, page)
    return render_template('personnel_administratif.html',
personnels=personnels_paginated, page=page, filtre_actif=filtre)

@app.route('/gestion_personnel/edit/<int:id_personnel>', methods=['GET'])
def show_edit_form(id_personnel):
    personnel = db.session.get(PERSONNEL, id_personnel)
    if not personnel:
        return "Personnel non trouvé", 404
    return render_template('edit_personnel.html', personnel=personnel)

@app.route('/gestion_personnel/edit/<int:id_personnel>', methods=['POST'])
@login_required
@role_access_rights(ROLE.administratif)
def edit_personnel(id_personnel):
    personnel = db.session.get(PERSONNEL, id_personnel)

    if personnel:
```

```
    try:
        personnel.nom = request.form.get('nom')
        personnel.prenom = request.form.get('prenom')
        personnel.role = request.form.get('role')
        new_mdp = request.form.get('mdp')
        if new_mdp:
            personnel.mdp = new_mdp

        db.session.commit()
    except Exception as e:
        db.session.rollback()
        print(f"Erreur lors de la mise à jour : {e}")

    return redirect(url_for('gestion_personnel'))
```

Implémentation des tests de la template :

```
@pytest.fixture
def testapp():
    app.config.update({"TESTING":True,"SQLALCHEMY_DATABASE_URI":
    'mysql://root:moins@127.0.0.1/LaboDino',"WTF_CSRF_ENABLED": False})
    yield app

@pytest.fixture
def client(testapp):
    return testapp.test_client()

def test_gestion_personnel_get_with_filter(client):
    """
    Test: -Filtrage par nom
        -Filtrage par prénom
        -Filtrage par rôle
    """
    client.post('/login/', data={'id': '16', 'password': 'MM@34'},
follow_redirects=True)

    response = client.get('/gestion_personnel/?filtre=nom')

    assert response.status_code == 200

    # Analyser le HTML de la réponse
    soup = BeautifulSoup(response.data, 'html.parser')

    # Trouver la première ligne (tr) dans le corps de la table (tbody)
    table_body = soup.find('table', {'id': 'table-admin'}).find('tbody')
    first_row = table_body.find('tr')

    # Extraire les cellules (td) de cette première ligne
    cells = first_row.find_all('td')
    # Vérifier le contenu de la première cellule (le nom)
    # text.strip(): nettoie le texte des espaces superflus
```

```
    first_personnel_nom = cells[0].text.strip()

    assert first_personnel_nom == 'Belobog'


    response = client.get('/gestion_personnel/?filtre=prenom')

    soup = BeautifulSoup(response.data, 'html.parser')

    table_body = soup.find('table', {'id': 'table-admin'}).find('tbody')
    first_row = table_body.find('tr')

    cells = first_row.find_all('td')
    first_personnel_prenom = cells[1].text.strip()

    assert first_personnel_prenom == 'Alexandrina'


    response = client.post('/gestion_personnel/', data={'filtre': 'role'})

    soup = BeautifulSoup(response.data, 'html.parser')

    table_body = soup.find('table', {'id': 'table-admin'}).find('tbody')
    first_row = table_body.find('tr')

    cells = first_row.find_all('td')
    first_personnel_prenom = cells[2].text.strip()

    assert first_personnel_prenom == 'administratif'
```

Conclusion :

Cette phase du projet a marqué une évolution significative dans mon approche du développement, passant de la simple implémentation fonctionnelle à la conception d'une application robuste et sécurisée.

La réalisation du module administratif a permis de valider des compétences techniques avancées :

Architecture et Données complexes (AC22.01) : La mise en place d'un ORM complexe avec SQLAlchemy (gestion des relations N-N et des énumérations) a assuré une structure de données fiable et cohérente et ainsi faire la connexion entre la vue et la base de données.

Sécurité et Contrôle (AC21.01) : L'implémentation de restrictions d'accès via des décorateurs personnalisés (@role_access_rights) démontre une maîtrise des enjeux de sécurité dans une application communicante en suivant la logique des exigences des opérations que.

Assurance Qualité (AC21.03) : L'aspect le plus marquant de cette partie est l'industrialisation des tests. L'utilisation combinée de coverage.py pour la couverture de code et de BeautifulSoup pour valider le rendu HTML montre l’adoption de bonnes pratique et le rendu d’une application fonctionnelle.

Enfin, l'intégration réussie de modules développés par d'autres membres de l'équipe (système de pagination) et la co-conception de la base de données illustrent ma capacité à collaborer efficacement (AC26.03) sur des briques techniques critiques. Cette partie 3 constitue ainsi le socle de stabilité nécessaire au déploiement de l'application.