



Licence MIAGE 2ème année

## Rapport de projet informatique

# Développement d'un site web et d'une base de données pour une association écologique avec assistance intelligente

Projet réalisé du Année universitaire 2025–2026 au

### Membres du groupe

**LYES YAHIAOUI – 45008773**  
**REDOUANE HAMMA – 45014843**

### Lien vers le dépôt Git

Le code source de l'application web ■ Nature ■ est accessible publiquement à l'adresse suivante :  
[https://github.com/lyesyahiooui/projet\\_informatique\\_12\\_univ\\_nanterre\\_25-26](https://github.com/lyesyahiooui/projet_informatique_12_univ_nanterre_25-26)

## **Remerciements**

Nous remercions l'équipe pédagogique de la licence 2 MIAGE ainsi que notre enseignant de projet informatique pour leur accompagnement et leurs conseils tout au long de ce travail.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Environnement de travail</b>	<b>4</b>
<b>3</b>	<b>Description du projet et objectifs</b>	<b>5</b>
3.1	Objectifs fonctionnels . . . . .	5
3.2	Objectifs techniques . . . . .	5
<b>4</b>	<b>Bibliothèques, Outils et technologies</b>	<b>6</b>
<b>5</b>	<b>Travail réalisé</b>	<b>7</b>
5.1	Liste des fonctionnalités prévues . . . . .	7
5.2	Fonctionnalités réalisées et non réalisées . . . . .	11
5.3	Répartition réelle du travail . . . . .	19
5.4	Utilisation de l'intelligence artificielle pendant le projet . . . . .	20
<b>6</b>	<b>Difficultés rencontrées</b>	<b>21</b>
<b>7</b>	<b>Bilan</b>	<b>21</b>
7.1	Conclusion . . . . .	21
7.2	Perspectives . . . . .	22
<b>8</b>	<b>Bibliographie</b>	<b>23</b>
<b>9</b>	<b>Webographie</b>	<b>24</b>
<b>10</b>	<b>Annexes</b>	<b>25</b>
<b>A</b>	<b>Cahier des charges</b>	<b>25</b>

# 1 Introduction

L'association ■ Nature et Environnement ■ est une structure universitaire qui organise des sorties thématiques afin de faire découvrir la faune, la flore et les milieux naturels aux adhérents. Elle s'inscrit dans une démarche de sensibilisation à la protection de l'environnement et de valorisation des espèces locales.

Dans ce contexte, le projet qui nous a été confié consiste à concevoir une base de données relationnelle cohérente, ainsi qu'un site web permettant à la fois de diffuser plus largement les informations sur les activités de l'association et de simplifier la gestion des adhérents et des sorties. Le site offre notamment la possibilité de consulter la liste des sorties proposées, de filtrer ces sorties selon différents critères (comme le thème, la date ou les places disponibles) et de s'y inscrire en ligne après authentification. Il met également à disposition un agent d'intelligence artificielle chargé de répondre aux questions fréquentes des adhérents, afin de les guider dans l'utilisation du site et la compréhension des fonctionnalités proposées. Les adhérents peuvent en outre accéder à un espace personnel où ils retrouvent l'historique des sorties auxquelles ils ont participé, ainsi que certaines statistiques liées à leurs activités.

Ce rapport présente tout d'abord l'environnement de travail utilisé, puis décrit le projet et ses objectifs, avant de détailler les principaux outils et technologies mis en œuvre. Il expose ensuite le travail réalisé et les difficultés rencontrées au cours du développement. Une attention particulière est portée aux fonctionnalités d'intelligence artificielle intégrées au site, en particulier un agent de FAQ combinant une recherche de similarité de textes et des appels à un modèle de langue, ainsi qu'à la manière dont l'IA nous a accompagnés dans la conception et la réalisation de ce projet.

# 2 Environnement de travail

Le développement du projet a été réalisé principalement sur un environnement GNU/Linux via le sous-système WSL (Windows Subsystem for Linux). Cet environnement nous a permis de disposer d'un système proche d'un serveur Linux classique pour l'exécution de l'application Flask et l'accès à la base de données PostgreSQL. L'édition du code a été effectuée avec l'environnement de développement Visual Studio Code, offrant une bonne intégration avec Python, Git et les outils de développement web.

La base de données relationnelle est gérée avec PostgreSQL, et l'application web y accède au moyen de la bibliothèque `psycopg2`, qui permet d'exécuter des requêtes SQL et de manipuler les résultats directement depuis le code Python. Le projet utilise également la bibliothèque `scikit-learn`, en particulier `TfidfVectorizer` et `cossine_similarity`, pour mettre en place une FAQ intelligente basée sur la similarité entre la question de l'utilisateur et une base de questions prédéfinies. Enfin, une API d'intelligence artificielle Mistral est intégrée pour générer des réponses complémentaires sur les thématiques liées à la nature et à l'environnement lorsque la FAQ ne suffit pas.

Le code source du projet est versionné avec Git et hébergé sur une plateforme GitHub, ce qui facilite la sauvegarde, le suivi des modifications et le partage du projet. Cette organisation a permis aux membres du groupe de collaborer efficacement, de tester progressivement les différentes fonctionnalités et d'assurer une meilleure traçabilité du développement.

### **3 Description du projet et objectifs**

Le projet vise à développer un site web connecté à une base de données relationnelle pour l'association ■ Nature et Environnement ■. Ce site doit permettre de centraliser les informations relatives aux adhérents et aux sorties, et d'offrir une interface conviviale pour consulter les activités proposées par l'association et s'y inscrire. Du côté des adhérents, l'application propose des pages publiques présentant les sorties, ainsi qu'un espace personnel accessible après authentification.

Le site permet notamment aux utilisateurs de consulter la liste des sorties proposées, d'en afficher les détails (thème, lieu, date, distance, effectif maximal, nombre d'inscrits) et de filtrer ces sorties selon différents critères comme le thème, la date ou le nombre de places restantes. Il propose également une page dédiée aux espèces, permettant aux adhérents de consulter la liste des espèces enregistrées dans la base de données et d'en visualiser les caractéristiques principales (nom, groupe, propriétés descriptives). Les adhérents peuvent en outre accéder à un espace personnel où ils retrouvent l'historique des sorties auxquelles ils ont participé, ainsi que certaines statistiques liées à leurs activités.

L'application intègre également un agent d'intelligence artificielle sous la forme d'une FAQ intelligente. Cet agent aide les adhérents à trouver rapidement des réponses aux questions fréquentes sur l'utilisation du site (création de compte, connexion, inscription à une sortie, consultation des espèces, etc.). Pour cela, le système combine une base de questions-réponses prédéfinies avec un mécanisme de recherche de similarité de textes, et peut, si nécessaire, faire appel à un modèle de langue Mistral pour fournir des réponses complémentaires sur les thématiques de nature et d'environnement. [file :3]

#### **3.1 Objectifs fonctionnels**

Les objectifs fonctionnels du projet sont les suivants :

- permettre aux visiteurs de consulter la liste des sorties organisées par l'association et d'en afficher les détails (thème, lieu, date, distance, effectif maximal, nombre d'inscrits) ;
- offrir aux adhérents un formulaire d'inscription et une page de connexion afin de créer et d'accéder à leur compte ;
- permettre aux adhérents authentifiés de s'inscrire à une sortie, avec vérification des places disponibles et de la date de la sortie ;
- fournir un espace adhérent affichant l'historique des sorties auxquelles l'utilisateur a participé, ainsi que des statistiques comme le nombre de sorties par mois ou la distance totale parcourue ;
- permettre la consultation d'un catalogue d'espèces, avec pour chaque espèce l'affichage de son nom, de son groupe et de ses principales caractéristiques ;
- proposer un agent de FAQ basé sur l'IA, capable de répondre aux questions fréquentes sur le fonctionnement du site et sur les activités de l'association.

#### **3.2 Objectifs techniques**

Sur le plan technique, le projet a pour objectifs :

- de concevoir un schéma de base de données relationnelle cohérent, modélisant les adhérents, les sorties et les inscriptions, avec des contraintes assurant l'intégrité des données ;

- de développer une application web en Python avec le framework Flask, organisée en routes claires pour l'accueil, la consultation des sorties, l'inscription, la connexion, l'espace adhérent et l'agent de FAQ ;
- d'intégrer PostgreSQL via la bibliothèque `psycopg2` pour exécuter des requêtes SQL avancées (jointures, agrégations, filtres, GROUP BY, HAVING) permettant de calculer par exemple le nombre de places restantes et les statistiques d'activité ;
- de mettre en œuvre une FAQ intelligente en utilisant `scikit-learn` (`TfidfVectorizer`, `cosine_similarity`) pour la recherche de similarité entre questions, et l'API Mistral pour générer des réponses contrôlées sur les thèmes de la nature et de l'environnement ;
- d'assurer une gestion correcte des sessions utilisateur (connexion, déconnexion, différenciation entre adhérent simple et gestionnaire) et de garantir un fonctionnement robuste de l'application.

## 4 Bibliothèques, Outils et technologies

-  **Flask** : framework web léger utilisé pour structurer l'application en différentes routes et gérer les interactions entre l'utilisateur, la logique métier et la base de données. Il permet de définir les pages du site (accueil, inscription, connexion, liste des sorties, détails d'une sortie, espace adhérent, FAQ IA) et de les associer à des gabarits HTML.
-  **PostgreSQL** : système de gestion de base de données relationnelle utilisé pour stocker les informations sur les adhérents, les sorties, les inscriptions et les espèces.
-  **psycopg2** : bibliothèque Python permettant de se connecter à PostgreSQL, d'exécuter des requêtes SQL (sélection, insertion, mise à jour) et de récupérer les résultats. Elle est au cœur de la communication entre Flask et la base de données.
-  **scikit-learn** : bibliothèque de machine learning utilisée pour la FAQ intelligente. Le composant `TfidfVectorizer` transforme les questions en vecteurs numériques, et `cosine_similarity` mesure la similarité entre la question de l'utilisateur et les questions de la FAQ afin de sélectionner la réponse la plus adaptée.
-  **API Mistral** : service d'intelligence artificielle utilisé pour générer des réponses lorsque la question ne correspond pas suffisamment aux entrées de la FAQ ou porte sur des aspects plus généraux liés à la nature et à l'environnement. Un prompt spécifique encadre le modèle afin de limiter les réponses au domaine du projet.
-  **VS Code et environnement WSL** : l'éditeur **Visual Studio Code**, associé à un environnement GNU/Linux via **WSL**, a été utilisé pour développer l'application, exécuter le serveur Flask, tester les requêtes SQL et gérer le projet avec Git et GitHub.

## 5 Travail réalisé

Dans cette section, nous précisons la liste des fonctionnalités prévues et distinguons celles qui ont été réalisées de celles qui ne l'ont pas été, en justifiant les choix et les éventuels abandons.

### 5.1 Liste des fonctionnalités prévues

#### Phase 1 : base de données et connexion

##### -1- Schéma Entité-Association

Dans une première étape, le travail a consisté à concevoir un schéma Entité-Association représentant les informations et relations décrites dans le cahier des charges : adhérents, sorties, inscriptions, domaines de spécialité, espèces locales, nichoirs, observations, cotisations et animateurs. Chaque entité du schéma possède ses attributs principaux (par exemple l'adhérent avec son identifiant, son nom, son adresse mail et son statut, ou la sortie avec son thème, son lieu de rendez-vous, sa date et sa distance), tandis que les associations modélisent la participation des adhérents aux sorties, la spécialisation dans certains domaines, la présence d'un animateur pour une sortie, le paiement des cotisations, la nidification des espèces dans les nichoirs et les observations réalisées lors des sorties.

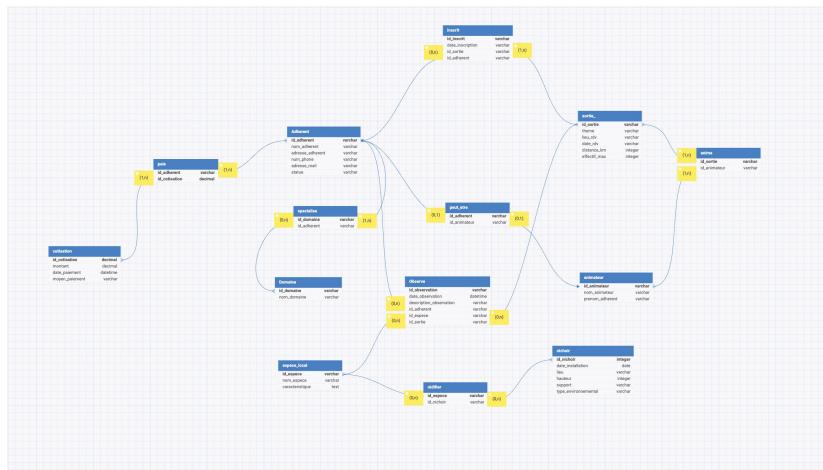


FIGURE 1 – Schéma Entité-Association du projet Nature et Environnement

##### -2- Crédit et peuplement de la base de données

Le schéma Entité-Association a ensuite été traduit en un schéma relationnel implanté dans PostgreSQL, décrit dans le fichier `dump.sql`. Chaque entité est devenue une table avec une clé primaire, et les associations ont été transformées en tables de liaison ou en tables portant leurs propres attributs. Par exemple, les tables `sortie`, `adhérent` et `gestionnaire` reprennent respectivement les informations sur les sorties proposées, les adhérents de l'association et les utilisateurs disposant de droits de gestion.

Une fois la structure de la base définie, des données de test ont été insérées afin de pouvoir vérifier le bon fonctionnement des requêtes SQL et des futures routes Flask. La table `sortie` est remplie avec plusieurs sorties thématiques comportant un thème,

```

1 CREATE TABLE sortie (
2     id_sortie SERIAL PRIMARY KEY,
3     theme VARCHAR(100) NOT NULL,
4     lieu_rdv VARCHAR(100) NOT NULL,
5     date_rdv TIMESTAMP NOT NULL,
6     dparcours INTEGER CHECK (dparcours > 0),
7     effectif_max INTEGER CHECK (effectif_max > 0)
8 );
9
10 CREATE TABLE adherent (
11     id_adh SERIAL PRIMARY KEY,
12     nom VARCHAR(50) NOT NULL,
13     prenom VARCHAR(50) NOT NULL,
14     adr_mail VARCHAR(100) UNIQUE NOT NULL,
15     num_phone VARCHAR(20) UNIQUE NOT NULL,
16     mot_de_passe VARCHAR(20) UNIQUE NOT NULL,
17     est_gestionnaire BOOLEAN DEFAULT FALSE
18 );
19 CREATE TABLE gestionnaire (
20     id_gestionnaire INTEGER NOT NULL UNIQUE REFERENCES adherent(id_adh) ON DELETE CASCADE ON UPDATE CASCADE
21 );
22
23

```

FIGURE 2 – Extrait de la création des tables principales dans `dump.sql`

```

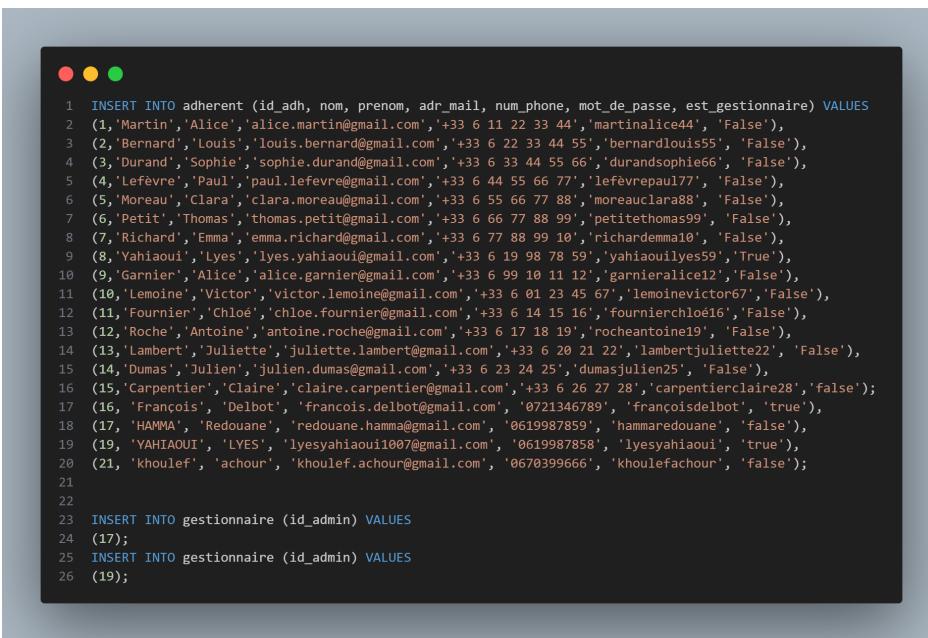
1 ---Filling table sortie :
2 INSERT INTO sortie (id_sortie, theme, lieu_rdv, date_rdv, dparcours, effectif_max) VALUES
3 (1, 'Observation des oiseaux migrateurs', 'Parc National', '2024-05-01 09:00:00', 5, 20),
4 (2, 'Randonnée ornithologique', 'Forêt de Fontainebleau', '2024-06-15 08:30:00', 10, 15),
5 (3, 'Atelier de nidification', 'Jardin Botanique', '2024-04-20 14:00:00', 3, 10),
6 (4, 'Sortie nocturne', 'Lac de Saint-Cassien', '2024-07-10 21:00:00', 4, 12),
7 (5, 'Observation des rapaces', 'Colline de la Croix-Rousse', '2024-08-05 07:00:00', 6, 25),
8 (6, 'Excursion en bateau', 'Lac Léman', '2024-09-12 10:00:00', 8, 30),
9 (7, 'Séance photo des oiseaux', 'Réserve Naturelle', '2024-10-01 16:00:00', 2, 18),
10 (8, 'Journée de nettoyage de la plage', 'Plage de Biarritz', '2024-11-15 09:30:00', 5, 25),
11 (9, 'Observation des oiseaux chanteurs', 'Parc de la Tête', '2024-12-20 08:00:00', 3, 20),
12 (10, 'Sortie familiale ornithologique', 'Zoo de Vincennes', '2024-05-15 10:30:00', 4, 40),
13 (11, 'Formation sur les espèces locales', 'Maison de la Nature', '2024-06-25 09:00:00', 2, 15),
14 (12, 'Atelier de construction de nichoirs', 'Centre Écologique', '2024-07-30 14:00:00', 3, 12),
15 (13, 'Sortie d\'observation à l\'aube', 'Réserve Ornithologique du Teich', '2024-08-20 06:30:00', 5, 20),
16 (14, 'Randonnée et observation des migrants', 'Montagne Sainte-Victoire', '2024-09-05 07:30:00', 7, 18),
17 (15, 'Sortie en kayak pour observer les oiseaux aquatiques', 'Rivière Dordogne', '2024-10-10 11:00:00', 6, 15),
18 (16, 'Randonnée sauvage', 'montagne Djurdjura en Kabylie', '2026-12-17 18:30:00', 180, 150),
19 (17, 'randonnée nanterrienne', 'ufr segmi', '2026-01-10 23:59:00', 99, 99999);

```

FIGURE 3 – Insertion de données d'exemple dans la table `sortie`

un lieu de rendez-vous, une date, une distance et un effectif maximal, ce qui permet de tester les filtres et le mécanisme d'inscription aux sorties.

De manière analogue, la table `adherent` est peuplée avec plusieurs adhérents fictifs, chacun disposant d'un identifiant, d'un nom, d'un prénom, d'une adresse mail, d'un numéro de téléphone, d'un mot de passe et d'un indicateur indiquant s'il s'agit ou non d'un gestionnaire. Certains de ces adhérents sont ensuite déclarés comme gestionnaires dans la table `gestionnaire`, ce qui permet de tester la distinction entre comptes simples et comptes ayant des droits d'administration dans l'application.



```

1  INSERT INTO adherent (id_adh, nom, prenom, adr_mail, num_phone, mot_de_passe, est_gestionnaire) VALUES
2  (1,'Martin','Alice','alice.martin@gmail.com','+33 6 11 22 33 44','martinalice44','False'),
3  (2,'Bernard','louis','louis.bernard@gmail.com','+33 6 22 33 44 55','bernardlouis55','False'),
4  (3,'Durand','Sophie','sophie.durand@gmail.com','+33 6 33 44 55 66','durandsophie66','False'),
5  (4,'Lefèvre','Paul','paul.lefevre@gmail.com','+33 6 44 55 66 77','lefevrepaul77','False'),
6  (5,'Moreau','Clara','clara.moreau@gmail.com','+33 6 55 66 77 88','moreauclarap88','False'),
7  (6,'Petit','Thomas','thomas.petit@gmail.com','+33 6 66 77 88 99','petitetthomas99','False'),
8  (7,'Richard','Emma','emma.richard@gmail.com','+33 6 77 88 99 10','richardemmal0','False'),
9  (8,'Yahiaoui','Lyes','lyes.yahiaoui@gmail.com','+33 6 19 98 78 59','yahiaouilyes59','True'),
10 (9,'Garnier','Alice','alice.garnier@gmail.com','+33 6 99 10 11 12','garnieralice12','False'),
11 (10,'Lemoine','Victor','victor.lemoine@gmail.com','+33 6 01 23 45 67','lemoinevictor67','False'),
12 (11,'Fournier','Chloé','chloe.fournier@gmail.com','+33 6 14 15 16','fournierchloé16','False'),
13 (12,'Roche','Antoine','antoine.roche@gmail.com','+33 6 17 18 19','rocheantoine19','False'),
14 (13,'Lambert','Juliette','juliette.lambert@gmail.com','+33 6 20 21 22','lambertjuliette22','False'),
15 (14,'Dumas','Julien','julien.dumas@gmail.com','+33 6 23 24 25','dumasjulien25','False'),
16 (15,'Carpentier','Claire','claire.carpentier@gmail.com','+33 6 26 27 28','carpentierclaire28','false'),
17 (16,'François','Delbot','francois.delbot@gmail.com','0721346789','francoisdelbot','true'),
18 (17,'HAMMA','Redouane','redouane.hamma@gmail.com','0619987859','hammaredouane','false'),
19 (19,'YAHIAOUI','LYES','lyesyahiaoui007@gmail.com','0619987858','lyesyahiaoui','true'),
20 (21,'khoulef','achour','khoulef.achour@gmail.com','0670399666','khoulefachour','false');
21
22
23 INSERT INTO gestionnaire (id_admin) VALUES
24 (17);
25 INSERT INTO gestionnaire (id_admin) VALUES
26 (19);

```

FIGURE 4 – Insertion de données d'exemple dans les tables `adherent` et `gestionnaire`

### -3- Module de connexion db.py

Afin de permettre à l'application Flask d'accéder à la base PostgreSQL `nature_db`, un module dédié `db.py` a été mis en place. Ce fichier centralise les paramètres de connexion (nom de la base, utilisateur, mot de passe, hôte, port) et définit une fonction `connect_to_db()` qui crée une connexion via la bibliothèque `psycopg2`. La connexion est configurée avec un curseur de type `NamedTupleCursor`, ce qui permet d'accéder aux colonnes par leur nom dans le code Python, et l'encodage `UTF8` est explicitement spécifié.



```
1 # -*- coding: utf-8 -*-
2 import psycopg2
3 import psycopg2.extras
4
5 def connect_to_db():
6     conn = psycopg2.connect(
7         dbname="nature_db",
8         user="lyesyahiaoui",
9         password="*Lyes200406#",
10        host="localhost",
11        port=5432,
12        cursor_factory=psycopg2.extras.NamedTupleCursor,
13        client_encoding='UTF8'
14    )
15
16    conn.autocommit = True
17    return conn
18
```

FIGURE 5 – Fonction `connect_to_db()` du module `db.py`

La fonction `connect_to_db()` active le mode `autocommit`, ce qui permet de valider automatiquement les requêtes d'insertion ou de mise à jour sans avoir à appeler explicitement `commit()` après chaque opération. Cette fonction est ensuite réutilisée dans les différentes routes de l'application Flask afin d'ouvrir une connexion vers la base de données, d'exécuter les requêtes SQL nécessaires (lecture, insertion, modification) puis de récupérer les résultats. Cette centralisation de la logique de connexion simplifie le code des routes et facilite une éventuelle modification future de la configuration de la base.

## Phase 2 : routes Flask et templates

L'application web est organisée autour de plusieurs routes Flask qui gèrent la navigation, l'authentification des adhérents et l'accès aux différentes fonctionnalités (inscription, connexion, consultation des sorties, espaces privés, agent FAQ). Chaque route récupère les données envoyées par l'utilisateur via des formulaires HTML, interagit avec la base PostgreSQL au moyen du module `db.py`, puis transmet les résultats à un template `Jinja2` chargé de l'affichage.

**Parcours d'inscription et de connexion** La route `/inscription` permet la création d'un nouveau compte adhérent. En méthode `POST`, elle vérifie que tous les champs du formulaire sont présents, récupère les informations saisies (nom, prénom, adresse mail, numéro de téléphone, mot de passe) puis ouvre une connexion via `db.connect_to_db()`. Elle exécute une requête `INSERT INTO adherent ... RETURNING id_adh` afin d'enregistrer l'adhérent et de récupérer son identifiant, qui est stocké dans la session pour connecter directement l'utilisateur à son espace. En cas de succès, un message de confirmation est affiché et l'utilisateur est redirigé vers son espace personnel.

### Phase 3 : agent d'IA pour la FAQ

- Mise en place d'une FAQ basée sur une base de questions-réponses prédéfinies.
- Utilisation de `TfidfVectorizer` et `cosine_similarity` de `scikit-learn` pour mesurer la similarité entre la question de l'utilisateur et les questions de la FAQ.
- Intégration d'un appel à l'API Mistral pour produire une réponse lorsque la similarité est insuffisante.
- Interface web permettant à l'utilisateur de poser une question à l'agent IA.

## 5.2 Fonctionnalités réalisées et non réalisées

### Phase 1 : base de données et connexion

#### Fonctionnalités réalisées :

- Schéma EA et modèle relationnel finalisé.
- Base PostgreSQL créée via `dump.sql`.
- Module `db.py` fonctionnel pour exécuter les requêtes SQL depuis Flask.

#### Fonctionnalités non réalisées / partielles et raisons :

- Certaines contraintes avancées (triggers, vérifications complexes sur les dates ou les capacités) n'ont pas été implémentées, par manque de temps et pour privilégier le développement des fonctionnalités visibles.

### Phase 2 : routes Flask et templates

L'application web est organisée autour de plusieurs routes Flask qui gèrent la navigation, l'authentification des adhérents et l'accès aux différentes fonctionnalités (inscription, connexion, consultation des sorties, espaces privés, agent FAQ). Chaque route récupère les données envoyées par l'utilisateur via des formulaires HTML, interagit avec la base PostgreSQL au moyen du module `db.py`, puis transmet les résultats à un template Jinja2 chargé de l'affichage.

**Parcours d'inscription et de connexion** La route `/inscription` permet la création d'un nouveau compte adhérent. En méthode `POST`, elle vérifie que tous les champs du formulaire sont présents, récupère les informations saisies (nom, prénom, adresse mail, numéro de téléphone, mot de passe) puis ouvre une connexion via `db.connect_to_db()`. Elle exécute une requête `INSERT INTO adherent ... RETURNING id_adh` afin d'enregistrer l'adhérent et de récupérer son identifiant, qui est stocké dans la session pour connecter directement l'utilisateur à son espace. En cas de succès, un message de confirmation est affiché et l'utilisateur est redirigé vers son espace personnel.

```

1  @app.route('/inscription', methods=['GET', 'POST'])
2  def inscription():
3      if request.method == 'POST':
4          # Vérifiez que tous les champs requis sont remplis
5          required_fields = ['nom', 'prenom', 'adr_mail', 'num_phone', 'mot_de_passe']
6          if not all(field in request.form for field in required_fields):
7              flash('Tous les champs sont requis', 'danger')
8              return render_template('page_inscription.html')
9
10         # Récupérer les données du formulaire
11         nom = request.form['nom']
12         prenom = request.form['prenom']
13         adr_mail = request.form['adr_mail']
14         num_phone = request.form['num_phone']
15         mot_de_passe = request.form['mot_de_passe']
16
17         # Connexion à la base de données
18         with db.connect_to_db() as conn:
19             try:
20                 with conn.cursor() as cur:
21                     # Insérer l'adhérent dans la base de données
22                     query = """
23                         INSERT INTO adherent (nom, prenom, adr_mail, num_phone, mot_de_passe)
24                         VALUES (%s, %s, %s, %s, %s)
25                         RETURNING id_adh;
26                     """
27                     cur.execute(query, (nom, prenom, adr_mail, num_phone, mot_de_passe))
28                     id_adh = cur.fetchone()[0] # Récupérer l'ID de l'adhérent nouvellement créé
29                     conn.commit() # Valider la transaction
30
31                     session["id_adherent"] = id_adh # Stocker l'ID dans la session
32                     flash('Inscription réussie !', 'success')
33                     return redirect(url_for('espace_adherent')) # Rediriger vers l'espace adhérent
34
35             except psycopg2.Error as e:
36                 flash(f'Erreur lors de l\'inscription : {str(e)}', 'danger')
37                 return render_template('page_inscription.html')
38
39         return render_template('page_inscription.html') # Affichez le formulaire d'inscription
40

```

FIGURE 6 – Route `/inscription` : création d'un nouvel adhérent

La route `/connexion` gère l'authentification. En POST, elle récupère l'adresse mail et le mot de passe saisis, puis interroge la table `adherent` pour vérifier la correspondance et récupérer le champ `est_gestionnaire`. Si les identifiants sont valides, l'identifiant de l'adhérent est stocké dans la session et, en fonction du rôle, l'utilisateur est redirigé soit vers l'espace adhérent, soit vers l'espace gestionnaire, avec un message de bienvenue. En cas d'erreur (identifiants incorrects ou problème de base de données), un message d'erreur est affiché et le formulaire de connexion est proposé à nouveau.



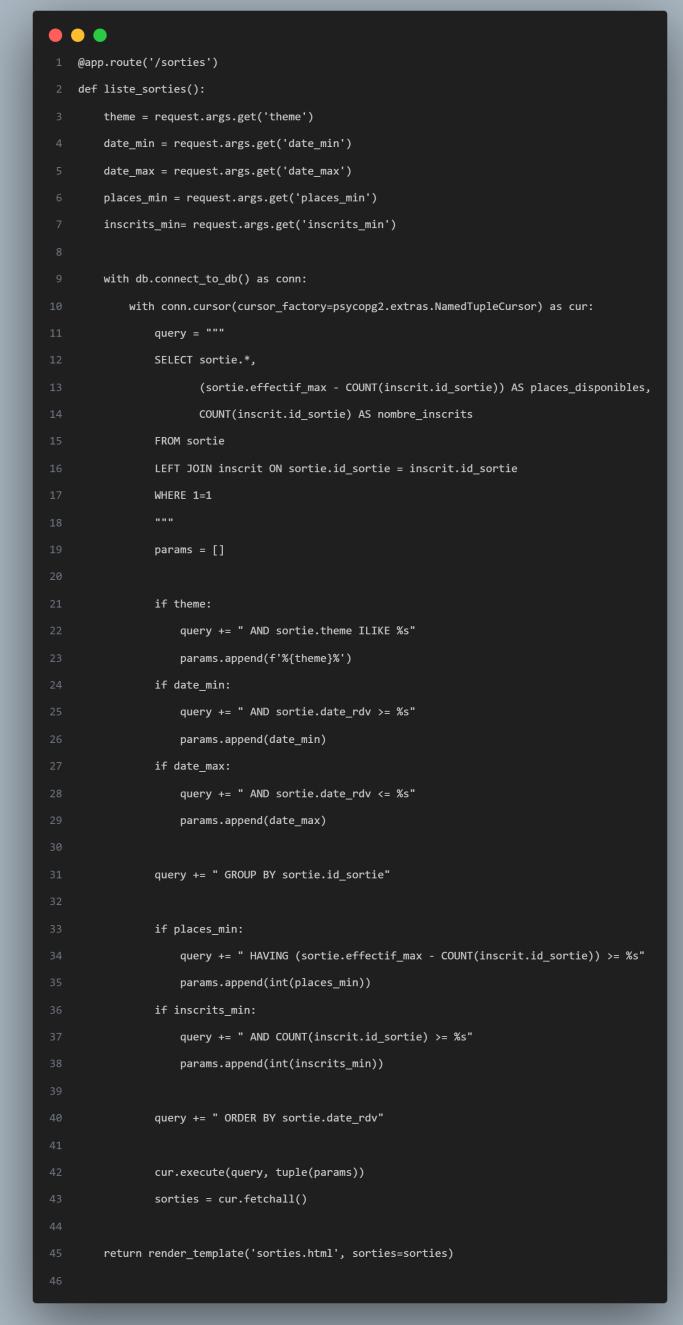
```

1  @app.route('/connexion', methods=['GET', 'POST'])
2  def connexion():
3      if request.method == 'POST':
4          # Récupérer les informations de connexion depuis le formulaire
5          email = request.form.get("adr_mail")           # name="adr_mail" dans connexion.html
6          password = request.form.get("mot_de_passe")    # name="mot_de_passe" dans connexion.html
7
8          with db.connect_to_db() as conn:
9              try:
10                  with conn.cursor() as cur:
11                      # Vérifier les identifiants + récupérer est_gestionnaire
12                      cur.execute("""
13                          SELECT id_adh, adr_mail, est_gestionnaire
14                          FROM adherent
15                          WHERE adr_mail = %s AND mot_de_passe = %s
16                      """ , (email, password))
17                      user = cur.fetchone()
18
19                  if user:
20                      id_adh, adr_mail, est_gestionnaire = user
21                      session["id_adherent"] = id_adh
22
23                      if est_gestionnaire:
24                          # Gestionnaire : accès à l'espace gestionnaire
25                          session["id_gestionnaire"] = id_adh
26                          flash('Bienvenue dans l'espace gestionnaire.', 'success')
27                          return redirect(url_for('espace_gestionnaire'))
28                      else:
29                          # Adhérent simple
30                          session.pop("id_gestionnaire", None)
31                          flash('Connexion réussie !', 'success')
32                          return redirect(url_for('espace_adherent'))
33                  else:
34                      flash('Identifiants incorrects.', 'danger')
35              except psycopg2.Error as e:
36                  flash(f'Erreur lors de la connexion : {str(e)}', 'danger')
37
38      # GET ou échec : afficher le formulaire de connexion
39      return render_template('connexion.html')
40

```

FIGURE 7 – Route /connexion : authentification des adhérents et des gestionnaires

**Gestion des sorties : liste, détails et inscription** La route /sorties liste les sorties proposées par l’association en tenant compte de plusieurs filtres optionnels passés en paramètres d’URL (thème, intervalle de dates, nombre minimal de places restantes, nombre minimal d’inscrits). Une requête SQL est construite dynamiquement à partir de ces filtres, en joignant la table sortie avec la table inscrit pour calculer, pour chaque sortie, le nombre de places disponibles et le nombre d’inscrits. Les résultats sont ensuite transmis au template sorties.html, qui affiche la liste des sorties et les contrôles de filtrage.



```

1  @app.route('/sorties')
2  def liste_sorties():
3      theme = request.args.get('theme')
4      date_min = request.args.get('date_min')
5      date_max = request.args.get('date_max')
6      places_min = request.args.get('places_min')
7      inscrits_min= request.args.get('inscrits_min')
8
9      with db.connect_to_db() as conn:
10         with conn.cursor(cursor_factory=psycopg2.extras.NamedTupleCursor) as cur:
11             query = """
12                 SELECT sortie.*,
13                     (sortie.effectif_max - COUNT(inscrit.id_sortie)) AS places_disponibles,
14                     COUNT(inscrit.id_sortie) AS nombre_inscrits
15                 FROM sortie
16                 LEFT JOIN inscrit ON sortie.id_sortie = inscrit.id_sortie
17                 WHERE 1=1
18                 """
19             params = []
20
21             if theme:
22                 query += " AND sortie.theme ILIKE %s"
23                 params.append(f'%{theme}%')
24             if date_min:
25                 query += " AND sortie.date_rdv >= %s"
26                 params.append(date_min)
27             if date_max:
28                 query += " AND sortie.date_rdv <= %s"
29                 params.append(date_max)
30
31             query += " GROUP BY sortie.id_sortie"
32
33             if places_min:
34                 query += " HAVING (sortie.effectif_max - COUNT(inscrit.id_sortie)) >= %s"
35                 params.append(int(places_min))
36             if inscrits_min:
37                 query += " AND COUNT(inscrit.id_sortie) >= %s"
38                 params.append(int(inscrits_min))
39
40             query += " ORDER BY sortie.date_rdv"
41
42             cur.execute(query, tuple(params))
43             sorties = cur.fetchall()
44
45     return render_template('sorties.html', sorties=sorties)
46

```

FIGURE 8 – Route /sorties : liste filtrable des sorties

Pour chaque sortie, la route /sorties/<int:id\_sortie> affiche une fiche de détail. Elle récupère toutes les informations de la sortie (thème, lieu, date, distance, effectif maximal) ainsi que le nombre d’inscrits et les places restantes via une requête SQL avec jointure et agrégation. Si l’identifiant demandé ne correspond à aucune sortie, un message d’erreur est affiché et l’utilisateur est redirigé vers la liste.

La route d’inscription à une sortie vérifie d’abord que l’utilisateur est connecté en contrôlant la présence de id\_adherent dans la session. Elle contrôle ensuite que l’adhérent n’est pas déjà inscrit à cette sortie et calcule le nombre de places restantes. Si des places sont encore disponibles, elle insère une nouvelle ligne dans la table inscrit avec la date du jour, puis affiche un message de succès ; sinon, un message signale que la sortie est complète.

```

1 @app.route('/sorties/<int:id_sortie>')
2 def details_sortie(id_sortie):
3     with db.connect_to_db() as conn:
4         with conn.cursor(cursor_factory=NamedTupleCursor) as cur:
5             # Récupérer les détails de la sortie
6             cur.execute("""
7                 SELECT sortie.*,
8                     (sortie.effectif_f_max - COUNT(inscrit.id_adh)) FILTER (WHERE inscrit.id_adh IS NOT NULL) AS places_disponibles,
9                     COUNT(inscrit.id_adh) AS nombre_inscrits
10                FROM sortie
11               LEFT JOIN inscrit ON sortie.id_sortie = inscrit.id_sortie
12              WHERE sortie.id_sortie = %s
13            GROUP BY sortie.id_sortie;
14             """, (id_sortie,))
15
16             sortie = cur.fetchone()
17
18     if sortie is None:
19         flash('Sortie non trouvée.', 'danger')
20     return redirect(url_for('liste_sorties'))
21
22 return render_template('details_sorties.html', sortie=sortie)
23

```

FIGURE 9 – Route d’inscription à une sortie : vérification des places disponibles et enregistrement

**Espaces privés pour les adhérents et les gestionnaires** L’espace adhérent, accessible via la route /espace-adherent, permet à un utilisateur connecté de consulter ses informations de base ainsi que l’historique des sorties auxquelles il a participé. La route commence par vérifier la présence de l’identifiant d’adhérent dans la session, puis récupère son nom et son prénom dans la table adherent. Une requête supplémentaire joint inscrit et sortie afin de lister les sorties passées, triées par date décroissante. Ces informations sont ensuite transmises au template espace\_adherent.html, qui affiche un espace personnalisé avec l’historique des activités.

```

1 @app.route('/espace-adherent')
2 def espace_adherent():
3     if 'id_adherent' in session: # Vérifier si l'utilisateur est connecté
4         with db.connect_to_db() as conn:
5             try:
6                 with conn.cursor(cursor_factory=psycopg2.extras.NamedTupleCursor) as cur:
7                     # Récupérer les informations de l'utilisateur connecté (nom et prénom)
8                     cur.execute("SELECT nom, prenom FROM adherent WHERE id_adh = %s", (session['id_adherent'],))
9                     user_info = cur.fetchone() # Récupérer les informations de l'utilisateur
10
11                 # Vérifier si l'utilisateur existe
12                 if user_info is None:
13                     flash('Utilisateur non trouvé.', 'danger')
14                     return redirect(url_for('connexion'))
15
16                 # Récupérer l'historique des sorties de l'adhérent
17                 query_historique = """
18                     SELECT sortie.theme, sortie.date_rdv, sortie.lieu_rdv
19                     FROM inscrit
20                     JOIN sortie ON inscrit.id_sortie = sortie.id_sortie
21                     WHERE inscrit.id_adh = %s
22                     ORDER BY sortie.date_rdv DESC;
23
24                 cur.execute(query_historique, (session['id_adherent'],))
25                 historique = cur.fetchall() # Récupérer toutes les sorties
26
27                 # Passer les données au template (historique, nom et prénom)
28                 return render_template('espace_adherent.html', historique=historique, nom=user_info.nom, prenom=user_info.prenom)
29             except psycopg2.Error:
30                 flash('Erreur lors de la récupération des données : {}'.format(str(e)), 'danger')
31                 return redirect(url_for('connexion')) # Rediriger vers la page de connexion en cas d'erreur
32
33             else:
34                 return redirect(url_for('connexion')) # Rediriger vers la page de connexion si non connecté
35

```

FIGURE 10 – Route /espace-adherent : historique des sorties d’un adhérent

L'espace gestionnaire, accessible via la route `/espace-gestionnaire`, est réservé aux utilisateurs disposant du rôle de gestionnaire. La route vérifie que `id_gestionnaire` est présent dans la session, puis récupère le nom et le prénom du gestionnaire dans la table `adherent`. Elle transmet ces informations au template `espace_gestionnaire.html`, qui constitue un point d'entrée vers les fonctionnalités d'administration (inscription d'adhérents, création de sorties, saisie de paiements, ajout d'espèces, etc.).



```

● ● ●
1 @app.route('/espace-gestionnaire')
2 def espace_gestionnaire():
3     if 'id_gestionnaire' not in session:
4         flash('Accès réservé aux gestionnaires.', 'danger')
5         return redirect(url_for('connexion'))
6
7     with db.connect_to_db() as conn:
8         try:
9             with conn.cursor(cursor_factory=psycopg2.extras.NamedTupleCursor) as cur:
10                 cur.execute("SELECT nom, prenom FROM adherent WHERE id_adh = %s", (session['id_gestionnaire'],))
11                 user_info = cur.fetchone()
12                 return render_template('espace_gestionnaire.html', nom=user_info.nom, prenom=user_info.prenom)
13             except psycopg2.Error as e:
14                 flash(f'Erreur lors de la récupération des données : {str(e)}', 'danger')
15             return redirect(url_for('connexion'))

```

FIGURE 11 – Route `/espace-gestionnaire` : accès aux fonctionnalités d'administration

### Phase 3 : agent d'IA pour la FAQ

Cette phase a consisté à concevoir et intégrer un agent d'intelligence artificielle dédié à une FAQ intelligente. L'objectif est d'aider les adhérents à obtenir rapidement des réponses aux questions fréquentes liées au site (inscription, connexion, sorties, espèces, etc.) tout en s'appuyant sur un modèle de langue pour les questions plus ouvertes sur la nature et l'environnement.

**Fonctions de réponse basées sur la base de données** Certains cas spécifiques sont traités directement à partir des données de la base, grâce à des fonctions Python dédiées. La fonction `reponse_especes_depuis_bdd` prend en paramètre une liste d'espèces (nom de l'espèce et nom du groupe) et construit une réponse textuelle courte présentant quelques espèces répertoriées. Elle limite le nombre d'espèces (`max_especes`) et la longueur totale de la réponse (`max_car`) afin de conserver un message lisible, puis ajoute une phrase invitant l'utilisateur à consulter la page ■ Espèces ■ pour la liste complète.

```

1 def reponse_especies_depuis_bdd(especes, max_especies=10, max_car=450):
2     """
3         Construit une réponse courte listant quelques espèces de l'association.
4         especes : rows avec (nom_espece, groupe_nom)
5         """
6         especes = list(especes[:max_especies])
7         if not especes:
8             return "Aucune espèce n'est actuellement enregistrée dans la base de données."
9
10        lignes = []
11        for e in especes:
12            lignes.append(f"- {e.nom_espece} ({e.groupe_nom})")
13        texte_temp = "\n".join(lignes)
14        if len(texte_temp) > max_car:
15            lignes.pop()
16            break
17
18    return (
19        "Voici quelques espèces répertoriées par l'association :\n"
20        + "\n".join(lignes)
21        + "\nTu peux voir la liste complète dans la page « Espèces » du site."
22    )
23
24

```

FIGURE 12 – Fonction `reponse_especies_depuis_bdd` : synthèse des espèces répertoriées

De manière similaire, la fonction `reponse_sorties_depuis_bdd` génère une réponse textuelle listant quelques sorties à venir à partir des champs thème, lieu de rendez-vous et date de rendez-vous. Elle parcourt les sorties renvoyées par la requête SQL, construit des lignes de la forme ■ - thème à lieu le JJ/MM/AAAA ■, et s'arrête dès que la taille maximale du texte est atteinte. La réponse se termine par un rappel indiquant que la page ■ Sorties ■ du site permet d'obtenir davantage de détails et de s'inscrire.

```

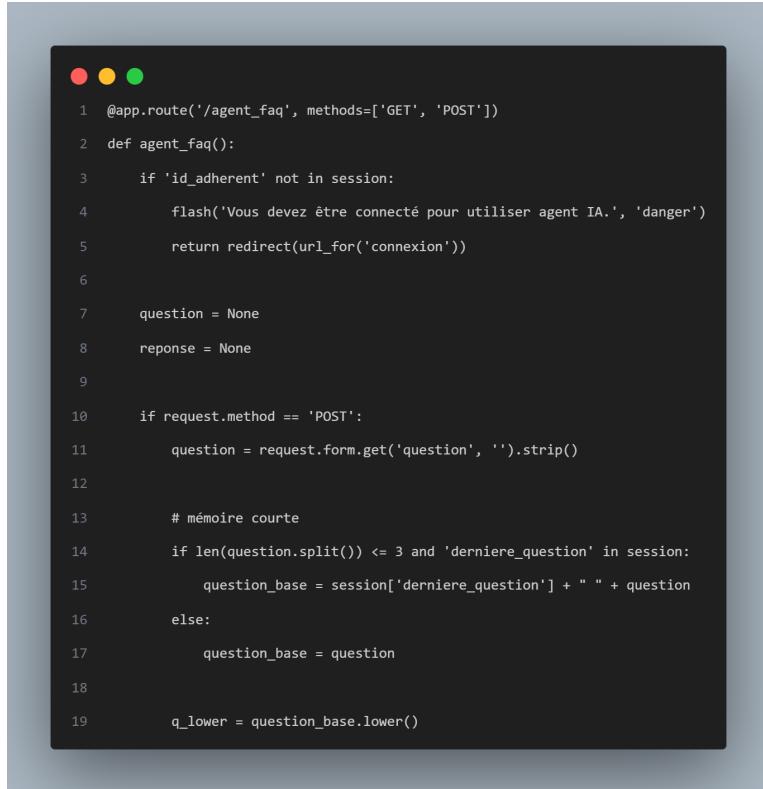
1 def reponse_sorties_depuis_bdd(sorties, max_sorties=10, max_car=450):
2     """
3         Construit une réponse courte listant quelques sorties de l'association.
4         sorties : rows avec (theme, lieu_rdv, date_rdv)
5         """
6         sorties = list(sorties[:max_sorties])
7         if not sorties:
8             return "Aucune sortie n'est actuellement enregistrée dans la base de données."
9
10        lignes = []
11        for s in sorties:
12            lignes.append(f"- {s.theme} à {s.lieu_rdv} le {s.date_rdv.strftime('%d/%m/%Y')}")
13        texte_temp = "\n".join(lignes)
14        if len(texte_temp) > max_car:
15            lignes.pop()
16            break
17
18    return (
19        "Voici quelques sorties actuellement proposées par l'association :\n"
20        + "\n".join(lignes)
21        + "\nConsulte la page « Sorties » du site pour plus de détails ou t'inscrire."
22    )
23
24

```

FIGURE 13 – Fonction `reponse_sorties_depuis_bdd` : synthèse des sorties proposées

**Route /agent\_faq et mémoire courte** L'agent IA est accessible via la route Flask `/agent_faq`, réservée aux adhérents connectés. Lorsqu'un utilisateur soumet une question, la route vérifie d'abord la présence de `id_adherent` dans la session, puis récupère le texte saisi dans le formulaire. Une logique de ■ mémoire courte ■ est ensuite appliquée : si la question courante est très brève (par exemple trois mots ou moins) et qu'une question plus détaillée a déjà été stockée dans `session['derniere_question']`, les deux sont concaténées afin de conserver le contexte de la conversation. La question

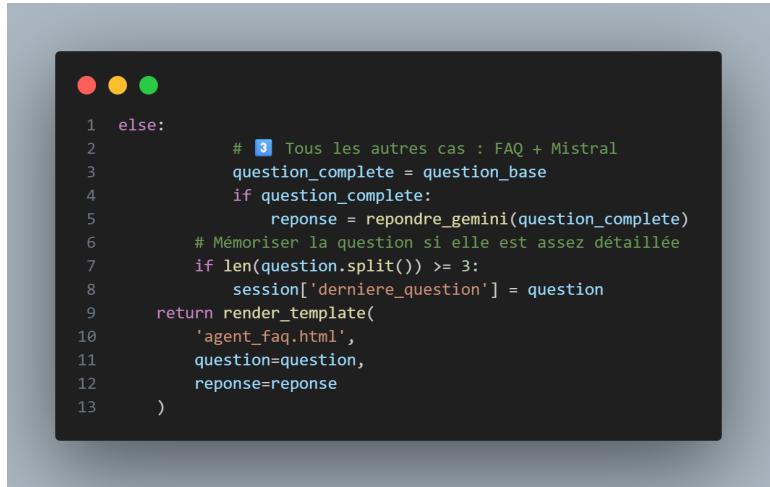
normalisée (`question_base`) est ensuite passée en minuscules pour faciliter les tests de présence de mots-clés.



```
 1 @app.route('/agent_faq', methods=['GET', 'POST'])
 2 def agent_faq():
 3     if 'id_adherent' not in session:
 4         flash('Vous devez être connecté pour utiliser agent IA.', 'danger')
 5         return redirect(url_for('connexion'))
 6
 7     question = None
 8     reponse = None
 9
10    if request.method == 'POST':
11        question = request.form.get('question', '').strip()
12
13        # mémoire courte
14        if len(question.split()) <= 3 and 'derniere_question' in session:
15            question_base = session['derniere_question'] + " " + question
16        else:
17            question_base = question
18
19        q_lower = question_base.lower()
```

FIGURE 14 – Début de la route `/agent_faq` : contrôle d'accès et mémoire courte

Deux cas spécifiques sont ensuite détectés à partir de mots-clés dans la question : si l'utilisateur demande la liste des sorties, une requête SQL est exécutée sur la table `sortie`, les résultats sont passés à `reponse_sorties_depuis_bdd` et la réponse générée est renvoyée. De même, si la question porte sur les espèces répertoriées, une requête est envoyée aux tables d'espèces et de groupes d'espèces, puis la fonction `reponse_especes_depuis_bdd` construit un texte de synthèse. Dans tous les autres cas, la route délégué le traitement à la fonction de FAQ intelligente `repondre_gemini` : la question complète est envoyée, une réponse est générée, et si la question est suffisamment détaillée, elle est enregistrée comme dernière question dans la session pour les interactions futures.



```

1  else:
2      # ❸ Tous les autres cas : FAQ + Mistral
3      question_complete = question_base
4      if question_complete:
5          reponse = repondre_gemini(question_complete)
6      # Mémoriser la question si elle est assez détaillée
7      if len(question.split()) >= 3:
8          session['derniere_question'] = question
9      return render_template(
10         'agent_faq.html',
11         question=question,
12         reponse=reponse
13     )

```

FIGURE 15 – Fin de la route /agent\_faq : appel au moteur de FAQ et mémorisation du contexte

**Combinaison FAQ / TF-IDF / modèle de langue** La fonction `repondre_gemini` constitue le cœur du moteur de FAQ intelligente. Elle commence par comparer la question à une base de questions fréquemment posées en utilisant `TfidfVectorizer` et la similarité cosinus : si la similarité entre la question et l'une des entrées de la FAQ dépasse un seuil, la réponse statique correspondante est renvoyée. Dans le cas contraire, la question est envoyée à un modèle de langue externe (Mistral) à travers un prompt spécialisé qui limite la réponse aux thématiques de la nature, de l'environnement, des espèces et des sorties. La réponse textuelle obtenue est finalement affichée dans la page `agent_faq.html`. Cette combinaison de requêtes SQL ciblées, de FAQ statique et d'IA générative permet d'offrir un agent d'aide à la fois pertinent, contrôlé et adapté au domaine du projet.

### 5.3 Répartition réelle du travail

La répartition des tâches s'est organisée autour des trois phases décrites précédemment.

**Phase 1 : base de données et connexion** Redouane a pris en charge la conception du schéma Entité-Association à partir du cahier des charges, en identifiant les entités principales (adhérents, sorties, inscriptions, espèces, nichoirs, animateurs, cotisations) ainsi que les associations et les cardinalités. [Nom ou toi] a ensuite traduit ce schéma en un modèle relationnel et rédigé le fichier `dump.sql` contenant les instructions de création de toutes les tables, avec les clés primaires, les clés étrangères et les contraintes d'intégrité. Lyes a développé le module `db.py` pour gérer la connexion à PostgreSQL et centraliser les appels à la base de données depuis l'application Flask. Les données de test ont été insérées conjointement pour vérifier le bon fonctionnement du schéma.

**Phase 2 : routes Flask et templates** LYES et Redouane ont développé l'ensemble des routes principales de l'application Flask : routes d'inscription et de connexion, routes de gestion des sorties (liste, filtres, détails, inscription), routes pour les espèces, routes pour les espaces adhérent et gestionnaire, et routes de gestion réservées aux administrateurs (création de sorties, inscription d'adhérents, etc.). [Nom si binôme]

s'est occupé de la création des templates HTML avec Jinja2 et de l'intégration du CSS pour la mise en page et la navigation. Les tests d'intégration (vérification des formulaires, des redirections et des messages d'erreur) ont été réalisés par les deux.

**Phase 3 : agent d'IA pour la FAQ** Cette phase a été entièrement prise en charge par Lyes . Elle a demandé la mise en place de la base de questions-réponses, l'intégration de `scikit-learn` pour la vectorisation TF-IDF et le calcul de similarité cosinus, et la connexion à l'API Mistral pour les questions hors FAQ. j'ai également ajouté la logique de mémoire courte et les cas spécifiques traités par requêtes SQL (sorties et espèces) pour enrichir l'agent. redouane a contribué aux tests de l'agent et à l'amélioration du prompt envoyé à l'IA.

**Rédaction du rapport** La rédaction du rapport (contexte, conception, travail réalisé, annexes) a été partagée entre moi et lui , avec une relecture croisée pour assurer la cohérence et la correction des parties. Redouane as rédigé les sections techniques (phases 1 et 3), tandis que Lyes s'est chargé de la présentation générale et des annexes.

## 5.4 Utilisation de l'intelligence artificielle pendant le projet

Au cours de ce projet, des outils d'intelligence artificielle générative ont été utilisés comme soutien ponctuel pour la conception de la base de données, le développement de l'application Flask et la mise en place de l'agent FAQ intelligent, sans jamais remplacer le travail de réflexion ni le codage réel effectué par les membres du groupe. Ils ont principalement servi à clarifier certains points techniques et à améliorer la présentation du rapport en L<sup>A</sup>T<sub>E</sub>X.

Pour la partie back-end, l'IA a aidé à structurer et vérifier la cohérence des routes Flask (`/inscription`, `/connexion`, `/sorties`, `/espace-adherent`, `/espace-gestionnaire`, `/agent_faq`) avec le schéma relationnel PostgreSQL mis en place. Elle a fourni des exemples de patrons de code pour la gestion des formulaires, des sessions, des messages `flash` et de l'utilisation de `psycopg2` dans le module `db.py`, que nous avons ensuite adaptés, complétés et testés sur notre propre structure de tables.

Pour l'agent FAQ, l'intelligence artificielle a contribué à définir l'architecture combinant FAQ statique, recherche de similarité TF-IDF avec `scikit-learn` et appel à un modèle de langue externe (API Mistral) lorsque la similarité est insuffisante. Elle a aussi aidé à concevoir la logique de ■ mémoire courte ■ dans la route `/agent_faq` et à formaliser les fonctions `reponse_sorties_depuis_bdd` et `reponse_especes_depuis_bdd`, qui construisent des réponses synthétiques à partir des tables `sortie` et des tables d'espèces.

Enfin, pour la rédaction du rapport, l'IA a été utilisée pour proposer une organisation claire des sections (■ Travail réalisé ■, ■ Phase 3 : agent d'IA pour la FAQ ■, ■ Répartition réelle du travail ■), suggérer des formulations plus précises et donner des conseils sur l'intégration des figures (captures d'écran des routes Flask et des fonctions Python) dans le document L<sup>A</sup>T<sub>E</sub>X. Tout le contenu proposé a été relu, corrigé et personnalisé par Lyes et Redouane afin de refléter fidèlement le fonctionnement réel du site web et la contribution de chacun.

## 6 Difficultés rencontrées

La réalisation de ce projet a été accompagnée de plusieurs difficultés techniques et organisationnelles. Une première source de complexité a concerné la conception du schéma de **base de données** et l'écriture des **requêtes SQL**. Il a fallu choisir une modélisation cohérente des adhérents, des sorties et des inscriptions, puis écrire des requêtes avec **jointures** et **agrégations** pour calculer par exemple le nombre d'inscrits ou le nombre de places disponibles pour chaque sortie. Certaines requêtes, notamment celles combinant des **GROUP BY** et des conditions sur les agrégats, ont demandé plusieurs essais et corrections avant de produire les résultats attendus.

La gestion de l'**authentification** et des **sessions utilisateur** a également posé des difficultés. Mettre en place un système de connexion fiable, stocker l'identifiant de l'adhérent dans la session, différencier l'affichage selon que l'utilisateur est connecté ou non, et gérer correctement la déconnexion ont nécessité une bonne compréhension du fonctionnement des sessions dans le framework **Flask**. Au début, certaines pages restaient accessibles alors que l'utilisateur n'était pas connecté, ou au contraire provoquaient des redirections inattendues. Ces problèmes ont été résolus en ajoutant des vérifications systématiques de la session et en clarifiant la logique de redirection dans chaque route.

L'intégration de l'**IA** a constitué une autre étape délicate. La mise en place de la FAQ intelligente a d'abord demandé de comprendre comment représenter les questions sous forme de vecteurs avec **TfidfVectorizer**, puis comment utiliser la **similarité cosinus** pour sélectionner la réponse la plus pertinente. Il a parfois fallu ajuster le seuil de similarité afin d'éviter que le système ne renvoie une réponse inadaptée quand la question de l'utilisateur était trop différente des questions prédéfinies. Une difficulté importante a également concerné la connexion avec l'**API** d'intelligence artificielle. La configuration de la clé d'API et l'envoi correct des requêtes ont demandé plusieurs essais, en particulier pour respecter le format attendu par le service et gérer les **erreurs de réponse**. La mise en place de la fonction **repondre\_gemini**, qui combine d'abord une recherche de similarité dans la FAQ puis un appel à l'API lorsque nécessaire, a été réalisée en s'inspirant d'exemples et grâce à l'aide des **des outils IA**, ce qui a permis de clarifier la structure du code et la façon de chaîner ces deux comportements.

Enfin, la coordination entre les différentes parties du projet (**base de données**, routes **Flask**, templates **HTML**, FAQ IA) a demandé une attention particulière. Une modification dans la structure de la base ou dans une route pouvait entraîner des erreurs dans les pages associées. Il a donc été nécessaire de tester régulièrement l'application de bout en bout et de corriger les incohérences au fur et à mesure. Ce travail de débogage et de vérification a pris du temps, mais il a permis de mieux structurer le code, de comprendre les liens entre les différents composants et de stabiliser progressivement l'ensemble du site.

## 7 Bilan

### 7.1 Conclusion

La réalisation de ce projet nous a permis de concevoir et de mettre en œuvre une application web complète répondant au besoin de l'association ■ Nature et Environnement ■. Le site développé centralise les informations sur les adhérents, les sorties et les espèces, et offre aux utilisateurs des fonctionnalités essentielles telles que la consultation

et le filtrage des sorties, l'inscription en ligne, l'accès à un espace personnel, ainsi que la consultation d'un catalogue d'espèces avec leurs caractéristiques (nom, groupe, propriétés descriptives). L'intégration d'une FAQ intelligente basée sur l'intelligence artificielle constitue également un apport important pour l'accompagnement des adhérents dans l'utilisation du site.

Sur le plan pédagogique, ce projet nous a permis de mobiliser et d'approfondir plusieurs compétences : modélisation de base de données relationnelle, écriture de requêtes SQL avancées, développement web avec Flask, gestion des sessions et intégration d'outils d'IA. Le résultat obtenu est une application fonctionnelle qui respecte les objectifs principaux du cahier des charges, tout en offrant une base solide pour de futures améliorations.

## 7.2 Perspectives

Plusieurs pistes d'amélioration peuvent être envisagées pour la suite du projet. Sur le plan fonctionnel, il serait possible d'enrichir l'interface utilisateur, de proposer davantage de filtres et de statistiques pour les adhérents, et d'ajouter de nouvelles pages dédiées par exemple aux actualités de l'association ou aux comptes rendus détaillés des sorties. Un système d'envoi de confirmation par email lors de l'inscription d'un nouvel adhérent ou lors de l'inscription à une sortie pourrait également améliorer l'expérience utilisateur en garantissant que les adhérents sont informés de leurs actions. La gestion des rôles pourrait être approfondie, en offrant des interfaces plus complètes pour les gestionnaires (gestion des animateurs, export de données, tableaux de bord).

La sécurité des données des utilisateurs constitue une priorité pour de futures versions. L'implémentation d'un système de communication sécurisé permettrait aux adhérents et aux gestionnaires de communiquer en temps réel. De plus, renforcer le chiffrement des mots de passe par un mécanisme de **hash** robuste (par exemple **bcrypt** ou **argon2**), ajouter une authentification multi-facteurs et mettre en place des mécanismes de prévention contre l'injection SQL (notamment par l'utilisation systématique de requêtes paramétrées) seraient des étapes essentielles pour garantir l'intégrité et la confidentialité des données.

Sur le plan technique, une modernisation de l'architecture serait possible en utilisant **Docker** et **Node.js** pour développer des templates plus performants et scalables, permettant une meilleure séparation des services et une déployabilité accrue. Cela faciliterait également le déploiement sur différents environnements (développement, test, production).

Concernant l'intelligence artificielle, des évolutions significatives consisteraient à améliorer l'agent de FAQ en lui permettant d'accepter d'autres formats d'entrée, notamment des **entrées vocales** pour offrir une meilleure accessibilité, ainsi que des **photos** pour des demandes d'identification d'espèces basées sur l'analyse d'images. Une meilleure gestion du contexte et un apprentissage continu à partir des questions réellement posées par les utilisateurs pourraient également améliorer la qualité des réponses. Des optimisations de performances, une meilleure gestion des erreurs et un déploiement sur un serveur en ligne pourraient enfin permettre de rapprocher davantage le projet des conditions réelles d'utilisation.

## 8 Bibliographie

[Grinberg18] M. Grinberg, *Flask Web Development : Developing Web Applications with Python*, O'Reilly Media, 2ème édition, 2018.

## 9 Webographie

[DOCFL] <https://flask.palletsprojects.com/> : documentation officielle de Flask, utilisée pour la structure de l'application web et la gestion des routes.

[POST] <https://www.postgresql.org/docs/> : documentation de PostgreSQL, utilisée pour la définition du schéma relationnel et l'écriture des requêtes SQL.

[DOCPG] <https://www.psycopg.org/docs/> : documentation de la bibliothèque psycopg2, utilisée pour la connexion à PostgreSQL et l'exécution des requêtes depuis Python.

[SKL] [https://scikit-learn.org/stable/modules/feature\\_extraction.html](https://scikit-learn.org/stable/modules/feature_extraction.html) : documentation de TfidfVectorizer et de la similarité cosinus, utilisée pour la mise en place de la FAQ intelligente. [file :3]

[MIS] <https://docs.mistral.ai/> : documentation de l'API Mistral, utilisée pour comprendre le format des requêtes et des réponses du modèle d'IA intégré au projet. [file :3]

[MDN] <https://developer.mozilla.org/fr/docs/Web> : documentation MDN, utilisée comme rappel pour le HTML, le CSS et certains aspects de JavaScript.

[W3S] <https://www.w3schools.com/> : tutoriels et exemples pour revoir les bases de la mise en forme en HTML/CSS.

[GH] <https://github.com/> : plateforme de gestion de code source utilisée pour héberger le dépôt du projet et son répertoire de rapport.

## **10 Annexes**

### **Annexe A : Cahier des charges**

Cette annexe présente une synthèse du cahier des charges du projet ■ Nature et Environnement ■. L'objectif principal est de définir une base de données relationnelle et un site web permettant à l'association de mieux gérer ses activités et de diffuser plus largement les informations sur la faune, la flore et les sorties organisées.

#### **-Gestion des adhérents**

Le système doit permettre de gérer les adhérents de l'association. Pour chaque adhérent, on enregistre au minimum son nom, son prénom et ses coordonnées (adresse mail, numéro de téléphone, etc.). Les adhérents peuvent disposer de différents statuts (étudiant, personnel de l'université, extérieur), et le site doit offrir des fonctionnalités d'inscription en ligne, de connexion et d'accès à un espace personnel. Le cahier des charges prévoyait également un suivi des cotisations et des paiements, mais cet aspect n'a pas été intégré dans la version actuelle de l'application.

#### **-Gestion des sorties**

De nombreuses sorties thématiques sont proposées aux adhérents. Pour chaque sortie, le système doit gérer au minimum le thème, le lieu et la date de rendez-vous, le nombre de kilomètres à parcourir, l'effectif maximal et la liste des inscrits. Un adhérent peut s'inscrire à autant de sorties qu'il le souhaite, mais ne peut pas s'inscrire à une sortie qui a déjà eu lieu ou à une sortie qui n'a plus de places disponibles. Le site web doit permettre de consulter la liste de toutes les sorties proposées, de filtrer cette liste par thème, places restantes, nombre d'inscrits ou dates, et d'offrir un mécanisme d'inscription en ligne dans la limite des places disponibles.

#### **-Répertoire des espèces locales**

Une des premières activités de l'association a été de répertorier les principales espèces locales de plantes et d'animaux. Le système doit donc comporter un module permettant d'enregistrer les espèces, de les regrouper par type (par exemple oiseaux, plantes, etc.) et de stocker diverses caractéristiques descriptives. Le site web sert d'encyclopédie des espèces répertoriées : les visiteurs peuvent consulter la liste des animaux et plantes observées par l'association, et le choix d'une espèce permet d'afficher les informations pertinentes à son sujet. La page des espèces doit également proposer une recherche par caractéristiques afin d'aider à identifier une espèce à partir de ses traits distinctifs.

#### **-Interface web et espace gestionnaire**

Le site web doit offrir une interface publique pour la consultation des sorties et des espèces, ainsi qu'un espace privé pour les adhérents et les gestionnaires. Un adhérent connecté doit pouvoir consulter l'historique des sorties auxquelles il a participé et accéder à différentes statistiques (par exemple le nombre de sorties par mois ou la distance totale parcourue). Une page privée est réservée aux gestionnaires de l'association

pour inscrire de nouveaux adhérents, créer de nouvelles sorties, gérer les informations affichées sur le site et administrer la base de données.

### -Agent d'IA pour la FAQ

En complément des exigences initiales, le projet prévoit la mise en place d'un agent d'intelligence artificielle dédié à une FAQ. Cet agent doit aider les adhérents à trouver rapidement des réponses aux questions fréquentes (création de compte, connexion, inscription à une sortie, consultation des espèces, etc.). Pour cela, le système s'appuie sur une base de questions-réponses prédéfinies combinée à une recherche de similarité de textes, et peut, si nécessaire, faire appel à un modèle de langue externe pour fournir des réponses complémentaires sur les thématiques liées à la nature et à l'environnement. Cette exigence vient renforcer l'aspect ergonomique du site et illustre l'intégration d'outils d'IA dans le projet.

## Annexe : Interfaces de l'application web

Cette annexe présente les principales interfaces du site web de l'association ■ Chez Nature ■.

### Page d'accueil

La page d'accueil présente le titre ■ Assistant Nature ■ et un menu de navigation (Accueil, Sorties, Espèces, Chatbot FAQ, Mon espace, Déconnexion).

## Inscription et connexion

L'utilisateur peut créer un compte adhérent en remplissant un formulaire (nom, prénom, e-mail, téléphone, mot de passe), puis se connecter via une page dédiée de saisie de l'adresse e-mail et du mot de passe.

### Inscription

Nom :

Prénom :

Email :

Numéro de téléphone :

Mot de passe :

© 2024 Chez Nature. Tous droits réservés.

### Connexion

Adresse e-mail :

Mot de passe :

© 2024 Chez Nature. Tous droits réservés.

## Pages des sorties et des espèces

La section ■ Sorties ■ permet de consulter la liste des sorties et le détail d'une sortie (date, lieu, nombre de places, bouton d'inscription).

# Détails de la Sortie

Accueil   Sorties   Espèces   Contact   Espace Adhérent   Déconnexion

**Sortie familiale ornithologique**

Date de RDV : 15/05/2024 10:30  
Lieu de RDV : Zoo de Vincennes  
Places disponibles : 39 / 40  
Nombre d'inscrits : 1

[S'inscrire](#)

**Actions supplémentaires**

[Retour à la liste des sorties](#)

© 2024 Chez Nature. Tous droits réservés.

La section ■ Espèces ■ donne accès aux espèces suivies par l'association (accès illustré via les réponses du chatbot).

## Assistant Nature

Accueil   Sorties   Espèces   Chatbot FAQ   Mon espace   Déconnexion

**Assistant Nature**  
En ligne - Répond instantanément

je veux visiter des animaux à Nanterre si c'est possible?

A l'instant

À Nanterre, tu peux observer des animaux dans le parc des Chanteraines (oiseaux, écureuils) ou à la ferme pédagogique de la Folie (animaux domestiques). Sinon, le parc André-Malraux abrite aussi une biodiversité intéressante. (Je peux seulement répondre sur la nature et l'environnement.)

A l'instant

Posez votre question ici... 

Comment adhérer ?   Les sorties   propositions des sorties   Observation

## Espace gestionnaire et formulaires d'ajout

L'■ Espace gestionnaire ■ offre un tableau de bord pour inscrire un adhérent, créer une sortie, saisir un paiement, ajouter une espèce ou un nichoir.

The screenshot shows the 'Espace Gestionnaire' homepage with a green header bar containing navigation links: 'Inscrire un adhérent', 'Créer une sortie', 'Saisir un paiement', 'Ajouter une espèce', 'Ajouter un nichoir', and 'Déconnexion'. Below the header is a large white central area with a rounded rectangle containing the text 'Bienvenue dans l'espace gestionnaire' and a bullet point '• Bienvenue dans l'espace gestionnaire.' At the bottom of this area is a green footer bar with the copyright notice '© 2024 Chez Nature. Tous droits réservés.'

Les écrans suivants illustrent les formulaires d'ajout et de gestion :

The first screenshot shows the 'Inscrire un Nouvel Adhérent' (Register a New Member) form. It contains fields for 'Nom' (Name), 'Prénom' (First Name), 'Adresse Email' (Email Address), 'Numéro de Téléphone' (Phone Number), and 'Mot de Passe' (Password). A green 'Inscrire' (Register) button is located at the bottom right. A small link 'Retour à l'Espace Gestionnaire' (Return to the Management Space) is visible at the bottom of the form area.

The second screenshot shows the 'Saisir un Paiement' (Enter a Payment) form. It includes fields for 'ID Adhérent' (Member ID), 'ID Statut' (Status ID), 'ID Cotisation' (Contribution ID), 'Moyen de paiement' (Payment Method), and 'Date de paiement' (Payment Date). A date input field uses the 'jj/mm/aaaa' format. A green 'Enregistrer le paiement' (Save the payment) button is at the bottom. Like the first form, it has a 'Retour à l'Espace Gestionnaire' link at the bottom.

## Ajouter un Nouveau Nichoir

Date d'installation :  
jj/mm/aaaa

Lieu d'installation :  
Exemple : Jardin de l'école

Observations :  
Nombre d'œufs :

Type d'occupation :  
Exemple : Nidification

Période de nidification :  
Exemple : Mars - Avril

**Ajouter le Nichoir**

[Retour à l'Espace Gestionnaire](#)

## Ajouter une Nouvelle Espèce

Nom de l'Espèce :

Groupe :

Caractéristiques :

- Alimentation variée
- Chant mélodieux
- Comportement grégaire
- Comportement territorial
- Espèce protégée
- Habitat varié
- Migration saisonnière
- Mœurs solitaires
- Nidification précoce
- Nourriture granivore
- Nourriture insectivore
- Nourriture omnivore
- Nourriture piscivore
- Plumage coloré
- Taille moyenne

**Ajouter l'Espèce**

[Retour à l'Espace Gestionnaire](#)

## Ajouter une Observation

ID Espèce :

ID Nichoir :

Nombre d'Oeufs :

Type d'Occupation :

Période de Nidification :

**Ajouter l'Observation**

© 2024 Chez Nature. Tous droits réservés.

## Chatbot : suggestions rapides

L'onglet ■ Chatbot FAQ ■ affiche l'assistant virtuel avec une zone de saisie de texte et des boutons de suggestions rapides (■ Comment adhérer ? ■, ■ Les sorties ■, ■ Propositions des sorties ■, ■ Observation ■).

## Assistant Nature

Assistant Nature  
En ligne - Répond instantanément

Accueil      Sorties      Espèces      Chatbot FAQ      Mon espace      Déconnexion

comment faire voir les espèces  
À l'instant

Les espèces répertoriées sont consultables dans la page dédiée aux espèces, qui affiche les animaux et plantes observés par l'association.  
À l'instant

Posez votre question ici... 

Comment adhérer ?      Les sorties      propositions des sorties      Observation

## Assistant Nature

The screenshot shows the Assistant Nature website with a green header bar. A message bubble from the bot says: "Pour devenir adhérent, utilisez le bouton « S'inscrire » sur la page d'accueil ou dans le menu, complétez le formulaire et validez. Votre compte est immédiatement actif." Below the message is a text input field with placeholder "Posez votre question ici..." and a green play button icon. At the bottom are four buttons: "Comment adhérer ?" (highlighted), "Les sorties", "propositions des sorties", and "Observation".

The screenshot shows the Assistant Nature website with a green header bar. A message bubble from the bot says: "Quelles sont les espèces proposées par l'association ?" Below the message is a text input field with placeholder "Posez votre question ici..." and a green play button icon. At the bottom are three buttons: "Comment adhérer ?" (highlighted), "Types de sorties", and "Espèces proposées".

## Chatbot : exemples de questions/réponses

Plusieurs captures montrent des échanges avec le chatbot : présentation de l'association, explication de la procédure d'adhésion, affichage des espèces, propositions de sorties nature et réponses sur les lieux d'observation d'animaux.

The screenshot shows the Assistant Nature website with a green header bar. A message bubble from the bot says: "Bonjour ! Je suis l'assistant virtuel de l'association. Posez-moi une question sur les adhésions, les sorties ou la nature !" Below the message is a text input field with placeholder "Posez votre question ici..." and a green play button icon. At the bottom are four buttons: "Comment adhérer ?" (highlighted), "Les sorties", "propositions des sorties", and "Observation".

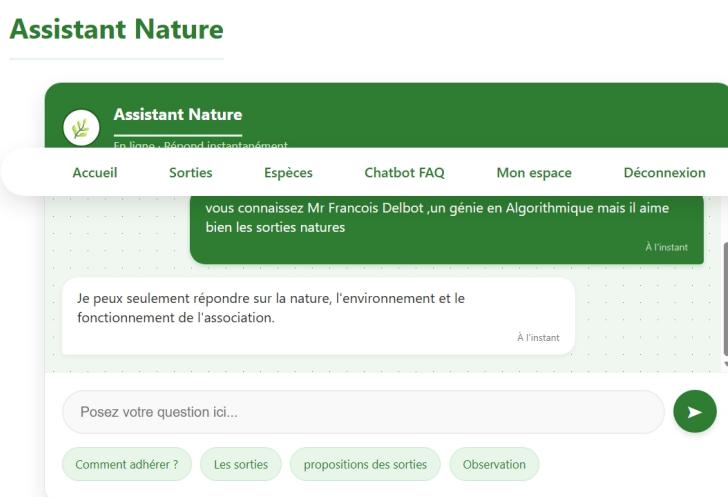
## Assistant Nature

The screenshot shows a chatbot interface for "Assistant Nature". At the top, there's a green header bar with the logo and the text "En ligne - Répond instantanément". Below the header are navigation links: Accueil, Sorties, Espèces, Chatbot FAQ, Mon espace, and Déconnexion. The main area is a conversation window. A message from the bot says "bonjour elle fait quoi votre association" with a timestamp "À l'instant". The user responds with "L'association Nature et environnement propose des sorties thématiques et des activités de protection et d'observation de la faune et de la flore locales." The bot replies with "À l'instant". Below the messages is a text input field "Posez votre question ici..." with a green send button. At the bottom, there are four buttons: "Comment adhérer?", "Les sorties", "propositions des sorties", and "Observation".

## Assistant Nature

This screenshot shows another conversation in the "Assistant Nature" chatbot. The bot asks "propose moi une sortie nature au sud de la france" with a timestamp "À l'instant". The user responds with "Découvrez les \*\*marais du Vigueirat\*\* (Bouches-du-Rhône) : balade en bateau pour observer flamants roses, hérons et biodiversité méditerranéenne. Idéal pour une sortie nature préservée." The bot replies with "À l'instant". Below the messages is a text input field "Posez votre question ici..." with a green send button. At the bottom, there are four buttons: "Comment adhérer?", "Les sorties", "propositions des sorties", and "Observation".

The final screenshot shows a third conversation. The bot asks "vous pensez quoi des oiseaux" with a timestamp "À l'instant". The user responds with "Les oiseaux sont essentiels aux écosystèmes : polliniseurs, disperseurs de graines et régulateurs des populations d'insectes. Leur déclin alerte sur la dégradation des habitats. Protégeons-les ! 🌸" The bot replies with "À l'instant". Below the messages is a text input field "Posez votre question ici..." with a green send button. At the bottom, there are four buttons: "Comment adhérer?", "Les sorties", "propositions des sorties", and "Observation".



## Manuel utilisateur

### 1. Connexion et inscription

Pour utiliser le site ■ Chez Nature ■, l'utilisateur ouvre l'application web dans un navigateur récent (Chrome, Firefox, Edge). Pour créer un compte, il remplit le formulaire d'inscription (nom, prénom, adresse e-mail, téléphone, mot de passe) puis clique sur ■ Incrire ■. Pour se connecter, il saisit son adresse e-mail et son mot de passe sur la page de connexion, puis valide avec le bouton ■ Se connecter ■.

### 2. Utilisation pour un adhérent

Une fois connecté, l'adhérent accède au menu principal ■ Accueil ■, ■ Sorties ■, ■ Espèces ■, ■ Chatbot FAQ ■, ■ Mon espace ■ et ■ Déconnexion ■. Dans l'onglet ■ Sorties ■, il consulte la liste des sorties, affiche le détail d'une sortie (date, lieu, nombre de places) et peut s'inscrire grâce au bouton ■ S'inscrire ■. Dans la section ■ Espèces ■, il visualise les espèces suivies par l'association ainsi que leurs principales caractéristiques.

### **3. Espace gestionnaire**

Les comptes disposant des droits de gestion accèdent à l’■ Espace gestionnaire ■. Cet espace permet d’inscrire un nouvel adhérent, de créer une sortie, d’enregistrer un paiement et de gérer la base naturaliste (ajout d’une espèce, d’un nichoir et des observations associées). Chaque fonctionnalité est accessible par un lien du menu et repose sur un formulaire dédié (saisie des champs obligatoires puis validation).

### **4. Chatbot ■ Assistant Nature ■**

L’onglet ■ Chatbot FAQ ■ donne accès à un assistant virtuel spécialisé sur la nature et le fonctionnement de l’association. L’utilisateur peut taper librement sa question ou cliquer sur les boutons de suggestion (■ Comment adhérer ? ■, ■ Les sorties ■, ■ Propositions des sorties ■, ■ Observation ■). Le chatbot renvoie une réponse textuelle expliquant, par exemple, la procédure d’adhésion, les sorties proposées, les espèces observées ou des idées de lieux pour observer la faune.

### **5. Déconnexion**

À la fin de la session, l’utilisateur clique sur ■ Déconnexion ■ afin de fermer sa session et de sécuriser son compte.