Réalisation du programme

EcoTrace

Calculateur d'Empreinte Carbone Personnelle

Mpia Mimpiya PULUDISU
Noméro étudiant : 18913467
L2 Informatique

15 août 2025

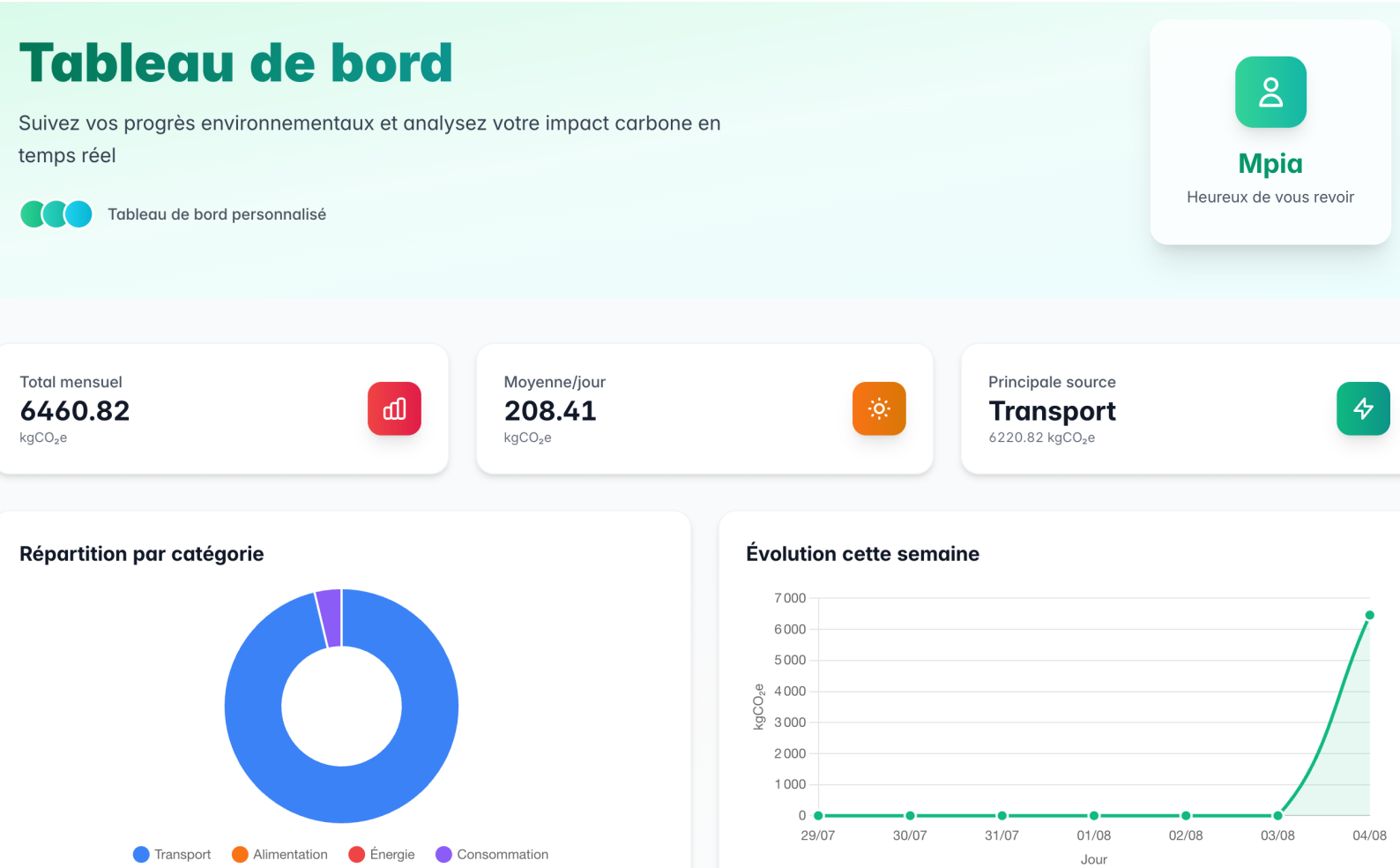


Table des matières

1 Attendus du projet EcoTrace

EcoTrace est une application web permettant aux utilisateurs de calculer, suivre et réduire leur empreinte carbone personnelle à travers l'enregistrement d'activités quotidiennes (transport, alimentation, consommation d'énergie). Ce projet répond à une préoccupation environnementale actuelle tout en s'inscrivant dans un périmètre adapté à une réalisation individuelle.

1.1 Fonctionnalités principales

1. **Calculateur d'empreinte carbone** :
   - Interface pour saisir les activités quotidiennes et leurs quantités
   - Base de données de facteurs d'émission pour différentes activités
   - Algorithme de calcul additif ($missions = quantit \times facteur\ d'mission$)
   - Calcul automatique de l'impact carbone en temps réel
2. **Suivi des progrès** :
   - Tableau de bord personnalisé avec statistiques en temps réel
   - Visualisations par jour/semaine/mois avec Chart.js
   - Répartition des émissions par catégorie (transport, alimentation, énergie, consommation)
   - Historique complet des activités avec possibilité de tri et filtrage
3. **Recommandations personnalisées** :
   - Moteur d'analyse automatique basé sur l'historique utilisateur
   - Suggestions adaptées au profil d'émission de l'utilisateur
   - Conseils pratiques pour réduire l'empreinte carbone
   - Système de filtrage par impact et facilité de mise en uvre

1.2 Technologies et méthodologie

1. Backend : Python avec Flask, architecture MVC et programmation orientée objet
   - Structure modulaire avec séparation des responsabilités
   - Classes CarbonCalculator et RecommendationEngine pour la logique métier
   - Gestion robuste des erreurs et validation des données
2. Frontend : HTML/CSS avec TailwindCSS, JavaScript moderne, Chart.js pour visualisations
   - Interface responsive et accessible
   - Composants interactifs (calendrier personnalisé, dropdowns)
   - Animations et micro-interactions pour l'engagement utilisateur
3. Base de données : SQLite avec SQLAlchemy (ORM)
   - Relations optimisées entre les tables
   - Méthodes de classe pour les statistiques globales
   - Initialisation automatique avec données ADEME
4. Source de données : Utilisation des données de l'ADEME (https://www.ademe.fr) pour les facteurs d'émission CO2
   - Facteurs d'émission officiels français
   - Couverture complète des activités quotidiennes
   - Données scientifiquement validées

TABLE 1 – Structure simplifiée des tables de données principales

Table	Champs
Users	id, name, email, password, is_admin, created_at
Activities	id, user_id, emission_factor_id, quantity, date, created_at
EmissionFactors	id, category, subcategory, activity_name, unit, co2_factor, source

1.3 Structure de données simplifiée

1.4 Réalisations techniques accomplies

1.4.1 Architecture backend

- **Module configs** : Configuration centralisée avec gestion d'erreurs et processeurs de contexte
- **Module controllers** : Classes CarbonCalculator et RecommendationEngine avec logique métier avancée
- **Module auth** : Système d'authentification complet avec Flask-Login et validation sécurisée
- **Module carbon** : Gestion des activités carbone avec validation multi-niveaux et cache LRU

1.4.2 Interface utilisateur

- **Templates responsive** : Design adaptatif avec TailwindCSS et composants réutilisables
- **JavaScript modulaire** : Scripts spécialisés pour chaque fonctionnalité avec optimisations
- **Visualisations** : Intégration Chart.js avec graphiques interactifs (doughnut, line, bar)
- **UX avancée** : Calendrier personnalisé, dropdowns stylisés, animations fluides

1.4.3 Déploiement et production

- **Déploiement réussi** : Application fonctionnelle sur PythonAnywhere
- **URL de production** : https://ecotrace.pythonanywhere.com
- **Code source** : https://github.com/codewithmpia/ecotrace-l2
- **Base de données initialisée** : Données ADEME intégrées et opérationnelles
- **Performance optimisée** : Minification automatique et lazy loading

1.5 Conformité aux attendus

Le projet EcoTrace répond intégralement aux exigences initiales et les dépasse sur plusieurs aspects :

- **Fonctionnalités réalisées** : Les 3 fonctionnalités principales sont implémentées et fonctionnelles
- **Technologies respectées** : Stack technique conforme aux spécifications
- **Structure de données** : Modèle de données optimisé et extensible
- **Qualité du code** : Architecture professionnelle avec bonnes pratiques
- **Déploiement effectif** : Application accessible publiquement

2 PARTIE I : Le module `configs`

2.1 Structure générale du module

Le module `configs` constitue le coeur de la configuration de l'application EcoTrace. Il est organisé en quatre fichiers principaux :

- `__init__.py` : Fichier d'initialisation du module (vide mais indispensable)
- `settings.py` : Configuration principale de l'application
- `errors.py` : Gestion centralisée des erreurs
- `processors.py` : Processeurs de contexte pour les templates

Cette organisation respecte le principe de séparation des responsabilités, rendant la configuration modulaire et facile à maintenir.

```
tree

.
├── __init__.py
├── errors.py
├── processors.py
└── settings.py

1 directory, 4 files
```

2.2 Configuration principale (`settings.py`)

Le fichier `settings.py` contient la configuration centrale de l'application. Voici son contenu :

```python
from pathlib import Path
import uuid

from flask import Flask, g
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, current_user
from flask_minify import Minify

BASE_DIR = Path(__file__).resolve().parent.parent

# Dossier de templates
TEMPLATE_DIR = str(BASE_DIR / "assets/templates")

# Dossier de statiques
STATIC_DIR = str(BASE_DIR / "assets/static")

# URI de la base de données
DB_URI = f"sqlite:///{BASE_DIR}/db.sqlite3"

# Création de l'application Flask
app = Flask(
    __name__,
    template_folder=TEMPLATE_DIR,
    static_folder=STATIC_DIR
)

# Clé secrète pour la session
app.config["SECRET_KEY"] = str(uuid.uuid4())

# Configuration de la base de données
app.config["SQLALCHEMY_DATABASE_URI"] = DB_URI
```

```
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db = SQLAlchemy(app)

# Initialisation de Flask-Minify
Minify(app=app, html=True, js=True, cssless=True)

# Context processors
from .processors import (
    inject_total_users,
    inject_total_activities,
    inject_get_total_emissions,
)

app.context_processor(inject_total_users)
app.context_processor(inject_total_activities)
app.context_processor(inject_get_total_emissions)


# Gestion des erreurs
from .errors import (
    page_not_found,
    internal_server_error
)
app.register_error_handler(404, page_not_found)
app.register_error_handler(500, internal_server_error)

# Enregistrement des applications
from auth.apps import auth
from carbon.apps import carbon

app.register_blueprint(auth, url_prefix="/auth")
app.register_blueprint(carbon, url_prefix="/")

from auth.models import User
from carbon.models import EmissionFactor

# Initialisation de la base de données
@app.before_request
def create_all():
    db.create_all()

    # Vérifier si les facteurs d'émission existent déjà
    if EmissionFactor.query.count() == 0:
        # Ajouter des facteurs d'émission par défaut
        default_factors = [
            # Transport
            EmissionFactor(
                category='transport',
                subcategory='voiture',
                activity_name='Voiture essence',
                unit='km',
                co2_factor=0.192,
                source='ADEME'
            ),
            EmissionFactor(
                category='transport',
                subcategory='voiture',
                activity_name='Voiture diesel',
                unit='km',
                co2_factor=0.173,
                source='ADEME'
```

```
        ),
        EmissionFactor(
            category='transport',
            subcategory='voiture',
            activity_name='Voiture électrique',
            unit='km',
            co2_factor=0.02,
            source='ADEME'
        ),
        EmissionFactor(
            category='transport',
            subcategory='transport en commun',
            activity_name='Bus',
            unit='km',
            co2_factor=0.103,
            source='ADEME'
        ),
        EmissionFactor(
            category='transport',
            subcategory='transport en commun',
            activity_name='Train',
            unit='km',
            co2_factor=0.0056,
            source='ADEME'
        ),
        EmissionFactor(
            category='transport',
            subcategory='avion',
            activity_name='Avion',
            unit='km',
            co2_factor=0.285,
            source='ADEME'
        ),

        # Alimentation
        EmissionFactor(
            category='food',
            subcategory='viande',
            activity_name='Boeuf',
            unit='kg',
            co2_factor=27.0,
            source='ADEME'
        ),
        EmissionFactor(
            category='food',
            subcategory='viande',
            activity_name='Poulet',
            unit='kg',
            co2_factor=5.15,
            source='ADEME'
        ),
        EmissionFactor(
            category='food',
            subcategory='viande',
            activity_name='Porc',
            unit='kg',
            co2_factor=5.8,
            source='ADEME'
        ),
        EmissionFactor(
            category='food',
```

```
            subcategory='produits laitiers',
            activity_name='Fromage',
            unit='kg',
            co2_factor=5.3,
            source='ADEME'
        ),
        EmissionFactor(
            category='food',
            subcategory='produits laitiers',
            activity_name='Lait',
            unit='L',
            co2_factor=0.94,
            source='ADEME'
        ),
        EmissionFactor(
            category='food',
            subcategory='légumes',
            activity_name='Légumes locaux de saison',
            unit='kg',
            co2_factor=0.5,
            source='ADEME'
        ),
        EmissionFactor(
            category='food',
            subcategory='légumes',
            activity_name='Légumes importés ou hors saison',
            unit='kg',
            co2_factor=2.7,
            source='ADEME'
        ),

        # Énergie
        EmissionFactor(
            category='energy',
            subcategory='électricité',
            activity_name='Électricité (mix français)',
            unit='kWh',
            co2_factor=0.057,
            source='ADEME'
        ),
        EmissionFactor(
            category='energy',
            subcategory='chauffage',
            activity_name='Gaz naturel',
            unit='kWh',
            co2_factor=0.205,
            source='ADEME'
        ),
        EmissionFactor(
            category='energy',
            subcategory='chauffage',
            activity_name='Fioul domestique',
            unit='L', co2_factor=3.25,
            source='ADEME'
        ),

        # Consommation
        EmissionFactor(
            category='consumption',
            subcategory='vêtements',
            activity_name='T-shirt',
```

```
                unit='unité',
                co2_factor=7.0,
                source='ADEME'
            ),
            EmissionFactor(
                category='consumption',
                subcategory='vêtements',
                activity_name='Jean',
                unit='unité',
                co2_factor=25.0,
                source='ADEME'
            ),
            EmissionFactor(
                category='consumption',
                subcategory='électronique',
                activity_name='Smartphone',
                unit='unité',
                co2_factor=80.0,
                source='ADEME'
            ),
            EmissionFactor(
                category='consumption',
                subcategory='électronique',
                activity_name='Ordinateur portable',
                unit='unité',
                co2_factor=156.0,
                source='ADEME'
            )
        ]

        db.session.bulk_save_objects(default_factors)
        db.session.commit()

# Configuration de Flask-Login
login_manager = LoginManager(app)
login_manager.login_view = "auth.login"
login_manager.login_message = "La connexion est requise."

@login_manager.user_loader
def load_user(id):
    return User.query.get(id)

@app.before_request
def get_current_user():
    g.user = current_user
```

Listing 1 – Code du module `settings.py` (configuration principale de l'application)

Les points clés de cette configuration sont :

1. **Gestion des chemins et de la base de données** :

```
BASE_DIR = Path(__file__).resolve().parent.parent
DB_URI = f"sqlite:///{BASE_DIR}/db.sqlite3"
```

   L'utilisation de SQLite assure la portabilité et la simplicité pour un projet individuel.

2. **Sécurité** :

```
app.config["SECRET_KEY"] = str(uuid.uuid4())
```

La clé secrète est générée dynamiquement à chaque démarrage pour renforcer la sécurité.

3. **Optimisation des performances** :
   Intégration de `FlaskMinify` pour optimiser automatiquement les ressources statiques (HTML, JS, CSS).
4. **Initialisation automatique des données** :

```
@app.before_request
def create_all():
    db.create_all()
    if EmissionFactor.query.count() == 0:
        # Insertion des facteurs d'émission par défaut
```

   Cela garantit que la base de données est toujours prête avec les données de référence issues de l'ADEME.
5. **Facteurs d'émission intégrés** :
   Les facteurs d'émission couvrent quatre grandes catégories :
   - *Transport* : voitures (essence, diesel, électrique), transports en commun, avion
   - *Alimentation* : viandes, produits laitiers, légumes
   - *Énergie* : électricité, chauffage (gaz, fioul)
   - *Consommation* : vêtements, électronique

   Toutes les données proviennent de l'ADEME pour garantir la fiabilité scientifique.
6. **Gestion de l'authentification** :
   Configuration de `FlaskLogin` pour :
   - Rediriger automatiquement vers la page de connexion
   - Charger automatiquement les utilisateurs
   - Gérer le contexte utilisateur global

2.3 Gestion des erreurs (`errors.py`)

Le module `errors.py` centralise la gestion des erreurs de l'application :

```python
from flask import flash, render_template


def get_form_errors(form):
    """
    Récupère les erreurs du formulaire et les affiche à l'utilisateur.
    """
    for _, errors in form.errors.items():
        for error in errors:
            flash(error, "danger")


def page_not_found(e):
    """
    Gère les erreurs 404.
    """
    ctx = {
        "title": "Page introuvable",
        e.description: "La page que vous recherchez n'existe pas.",
        "status": 404,
    }
    return render_template("errors/404.html", **ctx)
```

```python
def internal_server_error(e):
    """
    Gère les erreurs 500.
    """
    ctx = {
        "title": "Erreur serveur",
        e.description: "Une erreur interne s'est produite sur le serveur.",
        "status": 500,
    }
    return render_template("errors/500.html", **ctx)
```

Listing 2 – Code du module `errors.py` (gestion des erreurs Flask)

Il gère notamment :

- La fonction utilitaire `get_form_errors` pour l'affichage des erreurs de formulaire
- La gestion des erreurs HTTP (404, 500)
- L'utilisation de templates dédiés pour chaque type d'erreur

2.4 Processeurs de contexte (`processors.py`)

Le module `processors.py` contient trois processeurs de contexte qui enrichissent automatiquement tous les templates avec des statistiques globales :

```python
from auth.models import User
from carbon.models import Activity

def inject_total_users():
    return {"total_users": User.get_total_users()}


def inject_total_activities():
    return {"total_activities": Activity.get_total_activities()}

def inject_get_total_emissions():
    return {"total_emissions": User.get_all_users_total_emissions()}
```

Listing 3 – Code du module `processors.py` (processeurs de contexte Flask)

Les variables injectées sont :

- `inject_total_users` : nombre total d'utilisateurs inscrits
- `inject_total_activities` : nombre total d'activités enregistrées
- `inject_get_total_emissions` : émissions totales de tous les utilisateurs

Cette solution présente plusieurs avantages :

- **Automatisation** : Les données sont disponibles dans tous les templates sans intervention manuelle
- **Centralisation** : Évite la duplication de code
- **Cohérence** : Affichage uniforme des statistiques sur tout le site

À noter : ces fonctions s'exécutent à chaque requête, ce qui pourrait impacter les performances avec une base d'utilisateurs importante.

2.5 Conclusion

Le module `configs` constitue la base de l'application EcoTrace, assurant une configuration modulaire et maintenable. Il intègre les fonctionnalités essentielles pour le calcul de l'empreinte carbone, la gestion des erreurs et l'enrichissement des templates.

L'ensemble respecte les bonnes pratiques de développement en Python et Flask, tout en garantissant une expérience utilisateur fluide et cohérente.

3 PARTIE II : Le module `controllers`

Le module `controllers` gère la logique métier de l'application EcoTrace. Il est organisé autour de deux classes principales, chacune dans son propre fichier :

— `calculator.py` : contient la classe `CarbonCalculator` pour tous les calculs d'empreinte carbone.

— `recommendation.py` : contient la classe `RecommendationEngine` pour le système de recommandations personnalisées.

Cette séparation permet de distinguer clairement l'analyse des données (calculs) de l'accompagnement utilisateur (recommandations).

```
~/Documents/GitHub/ecotrace-l2/controllers git:(main)  5 files c
tree

.
├── calculator.py
└── recommendation.py

1 directory, 2 files
```

3.1 Classe `CarbonCalculator` (`calculator.py`)

Cette classe centralise tous les calculs relatifs à l'empreinte carbone d'un utilisateur. Voici son contenu :

```python
# controllers/calculator.py
from datetime import datetime, timedelta
import calendar

from carbon.models import Activity, EmissionFactor


class CarbonCalculator:
    """Classe pour calculer l'empreinte carbone des utilisateurs"""

    def __init__(self, user_id):
        """Initialisation avec l'ID utilisateur"""
        self.user_id = user_id

    def calculate_daily_footprint(self, date):
        """
        Calcule l'empreinte carbone pour un jour spécifique

        Args:
            date: Date pour laquelle calculer l'empreinte

        Returns:
            dict: Empreinte totale et répartition par catégorie
        """
        # Récupérer toutes les activités de l'utilisateur pour ce jour
        activities = Activity.query.filter_by(
            user_id=self.user_id,
            date=date
        ).all()

        # Initialiser les compteurs
        total_emissions = 0
        by_category = {
            'transport': 0,
            'food': 0,
```

```
            'energy': 0,
            'consumption': 0
        }

        # Calculer les émissions pour chaque activité
        for activity in activities:
            # Vérifier si emission_factor_id existe
            if activity.emission_factor_id is None:
                continue

            # Récupérer le facteur d'émission
            emission_factor = EmissionFactor.query.get(activity.
               emission_factor_id)

            # Vérifier si le facteur d'émission existe
            if emission_factor is None:
                continue

            # Vérifier si la catégorie est valide
            if emission_factor.category not in by_category:
                continue

            # Calculer les émissions
            try:
                emissions = activity.quantity * emission_factor.co2_factor
            except Exception as e:
                continue

            # Ajouter au total
            total_emissions += emissions

            # Ajouter à la catégorie correspondante
            by_category[emission_factor.category] += emissions

        # Arrondir les valeurs pour l'affichage
        for category in by_category:
            by_category[category] = round(by_category[category], 2)

        return {
            'total': total_emissions,
            'by_category': by_category
        }

    def calculate_weekly_trend(self):
        """
        Calcule la tendance des émissions sur les 7 derniers jours

        Returns:
            list: Liste de dictionnaires avec la date et les émissions
        """
        today = datetime.now().date()
        start_date = today - timedelta(days=6)  # 7 jours au total

        # Liste pour stocker les résultats
        daily_emissions = []

        # Calculer les émissions pour chaque jour
        current_date = start_date

        while current_date <= today:
            result = self.calculate_daily_footprint(current_date)
```

```
            daily_emissions.append({
                'date': current_date.strftime('%Y-%m-%d'),
                'display_date': current_date.strftime('%d/%m'),
                'emissions': round(result['total'], 2)
            })
            current_date += timedelta(days=1)

        return daily_emissions

    def calculate_monthly_summary(self, month=None, year=None):
        """
        Calcule le résumé des émissions pour un mois donné

        Args:
            month: Mois (1-12, par défaut mois actuel)
            year: Année (par défaut année actuelle)

        Returns:
            dict: Résumé des émissions pour le mois
        """
        # Si mois/année non spécifiés, utiliser le mois actuel
        if month is None or year is None:
            now = datetime.now()
            month = month or now.month
            year = year or now.year

        # Déterminer le premier et le dernier jour du mois
        first_day = datetime(year, month, 1).date()
        last_day = datetime(year, month, calendar.monthrange(year, month)[1]).
            date()

        # Récupérer toutes les activités du mois
        activities = Activity.query.filter(
            Activity.user_id == self.user_id,
            Activity.date >= first_day,
            Activity.date <= last_day
        ).all()

        # Initialiser les compteurs
        total_emissions = 0
        by_category = {
            'transport': 0,
            'food': 0,
            'energy': 0,
            'consumption': 0
        }

        # Calculer les émissions pour chaque activité
        for activity in activities:
            emission_factor = EmissionFactor.query.get(activity.
                emission_factor_id)
            emissions = activity.quantity * emission_factor.co2_factor

            # Ajouter au total
            total_emissions += emissions

            # Ajouter à la catégorie correspondante
            by_category[emission_factor.category] += emissions

        # Calculer la moyenne journalière
        days_in_month = (last_day - first_day).days + 1
```

```
        daily_average = total_emissions / days_in_month if days_in_month > 0
            else 0

        return {
            'total': round(total_emissions, 2),
            'daily_average': round(daily_average, 2),
            'by_category': {cat: round(val, 2) for cat, val in by_category.
                items()}
        }

    def compare_with_average(self):
        """
        Compare l'empreinte de l'utilisateur avec la moyenne nationale

        Returns:
            dict: Comparaison en pourcentage
        """
        # Moyenne nationale (en kg CO2 par jour) - valeurs fictives pour l'
            exemple
        national_average = {
            'total': 12.0,  # Moyenne totale
            'transport': 5.0,
            'food': 3.5,
            'energy': 2.0,
            'consumption': 1.5
        }

        # Calculer l'empreinte de l'utilisateur pour aujourd'hui
        today = datetime.now().date()
        user_footprint = self.calculate_daily_footprint(today)

        # Calculer le pourcentage par rapport à la moyenne
        comparison = {}

        # Total
        if national_average['total'] > 0:
            comparison['total'] = (user_footprint['total'] / national_average[
                'total']) * 100
        else:
            comparison['total'] = 0

        # Par catégorie
        comparison['by_category'] = {}

        for category in user_footprint['by_category']:
            if national_average[category] > 0:
                comparison['by_category'][category] = (user_footprint['
                    by_category'][category] / national_average[category]) * 100
            else:
                comparison['by_category'][category] = 0

        return {
            'national_average': national_average,
            'user_footprint': user_footprint,
            'percentage': {k: round(v, 1) for k, v in comparison.items() if k
                != 'by_category'},
            'percentage_by_category': {k: round(v, 1) for k, v in comparison['
                by_category'].items()}
        }
```

Listing 4 – Code du module `calculator.py` (calcul de l'empreinte carbone)

La classe `CarbonCalculator` est conçue pour être instanciée par utilisateur, ce qui facilite la gestion des données personnalisées :

```
class CarbonCalculator:
    def __init__(self, user_id):
        self.user_id = user_id
```

Les principales méthodes de cette classe sont :

1. **`calculate_daily_footprint`** :
   Calcule l'empreinte carbone pour une journée donnée, avec gestion robuste des erreurs et répartition par catégorie (transport, alimentation, énergie, consommation).

   ```
   if activity.emission_factor_id is None:
       continue
   if emission_factor is None:
       continue
   if emission_factor.category not in by_category:
       continue
   ```

2. **`calculate_weekly_trend`** :
   Fournit une vision temporelle sur 7 jours, avec des données formatées pour l'affichage graphique.

   ```
   def calculate_weekly_trend(self):
       today = datetime.now().date()
       start_date = today - timedelta(days=6)  # 7 jours au total
   ```

3. **`calculate_monthly_summary`** :
   Calcule un résumé mensuel : émissions totales, moyenne journalière, répartition par catégorie. Utilise `calendar.monthrange()` pour gérer la longueur variable des mois.
4. **`compare_with_average`** :
   Compare les émissions de l'utilisateur avec des moyennes nationales (valeurs de référence fixes pour le prototype, extensibles via API ou base de données).

3.2 Classe `RecommendationEngine` (`recommendation.py`)

Cette classe gère la génération de recommandations personnalisées pour aider l'utilisateur à réduire son empreinte carbone. Voici son contenu :

```
# controllers/recommendation.py
from datetime import datetime, timedelta
from carbon.models import Activity, EmissionFactor


class RecommendationEngine:
    """Moteur de recommandations pour réduire l'empreinte carbone"""

    def __init__(self, user_id):
        """Initialisation avec l'ID utilisateur"""
        self.user_id = user_id

    def get_personalized_recommendations(self):
        """
        Génère des recommandations personnalisées basées sur l'historique de l
            'utilisateur
```

```
        Returns:
            list: Liste de recommandations
        """
        try:
            # Récupérer les activités récentes (30 derniers jours)
            today = datetime.now().date()
            start_date = today - timedelta(days=30)

            recent_activities = Activity.query.filter(
                Activity.user_id == self.user_id,
                Activity.date >= start_date
            ).all()

            # Si pas assez de données, retourner des recommandations géné
               riques
            if len(recent_activities) < 5:
                return self._get_generic_recommendations()

            # Analyser les données pour identifier les domaines d'amélioration
            emissions_by_category = {
                'transport': 0,
                'food': 0,
                'energy': 0,
                'consumption': 0
            }

            # Calculer les émissions par catégorie
            for activity in recent_activities:
                # Vérifier si emission_factor_id existe et est valide
                if activity.emission_factor_id is None:
                    continue

                # Récupérer le facteur d'émission
                emission_factor = EmissionFactor.query.get(activity.
                   emission_factor_id)

                # Vérifier si le facteur d'émission existe
                if emission_factor is None:
                    continue

                # Vérifier si la catégorie est valide
                if emission_factor.category not in emissions_by_category:
                    continue

                # Calculer les émissions
                try:
                    emissions = activity.quantity * emission_factor.co2_factor
                    emissions_by_category[emission_factor.category] +=
                       emissions

                except Exception as e:
                    continue

            # Identifier la catégorie avec le plus d'émissions
            if all(value == 0 for value in emissions_by_category.values()):
                # Si toutes les catégories sont à 0, retourner des
                   recommandations génériques
                return self._get_generic_recommendations()

            max_category = max(emissions_by_category.items(), key=lambda x: x
               [1])[0]
```

```
            # Générer des recommandations spécifiques pour cette catégorie
            category_recommendations = self._get_category_recommendations(
                max_category)

            # Ajouter quelques recommandations génériques
            generic_recommendations = self._get_generic_recommendations()[:2]

            # Combiner les recommandations
            all_recommendations = category_recommendations +
                generic_recommendations

            return all_recommendations

        except Exception as e:
            # En cas d'erreur, retourner des recommandations génériques
            return self._get_generic_recommendations()

    def _get_category_recommendations(self, category):
        """
        Génère des recommandations spécifiques à une catégorie

        Args:
            category: Catégorie pour laquelle générer des recommandations

        Returns:
            list: Liste de recommandations pour cette catégorie
        """
        recommendations = []

        if category == 'transport':
            recommendations = [
                {
                    'title': 'Réduisez vos déplacements en voiture',
                    'description': 'Essayez de combiner plusieurs courses en
                        un seul trajet pour réduire votre kilométrage
                        hebdomadaire.',
                    'impact': 'Moyen',
                    'ease': 'Facile'
                },
                {
                    'title': 'Utilisez plus les transports en commun',
                    'description': 'Remplacer un trajet en voiture par les
                        transports en commun peut réduire vos émissions de CO2
                        jusqu\'à 80%.',
                    'impact': 'Élevé',
                    'ease': 'Moyen'
                },
                {
                    'title': 'Essayez le covoiturage',
                    'description': 'Partager votre trajet avec d\'autres
                        personnes divise les émissions par le nombre de
                        passagers.',
                    'impact': 'Élevé',
                    'ease': 'Moyen'
                }
            ]
        elif category == 'food':
            recommendations = [
                {
                    'title': 'Réduisez votre consommation de viande rouge',
```

```
                'description': 'Remplacer la viande rouge par de la
                   volaille peut réduire vos émissions alimentaires de
                   plus de 70%.',
                'impact': 'Très élevé',
                'ease': 'Moyen'
            },
            {
                'title': 'Privilégiez les produits locaux et de saison',
                'description': 'Les légumes importés ou hors saison
                   peuvent émettre jusqu\'à 5 fois plus de CO2 que les
                   produits locaux de saison.',
                'impact': 'Moyen',
                'ease': 'Facile'
            },
            {
                'title': 'Réduisez le gaspillage alimentaire',
                'description': 'Planifiez vos repas et utilisez les restes
                    pour éviter de jeter de la nourriture.',
                'impact': 'Moyen',
                'ease': 'Facile'
            }
        ]
    elif category == 'energy':
        recommendations = [
            {
                'title': 'Réduisez votre chauffage de 1řC',
                'description': 'Baisser la température de votre logement
                   de 1řC peut réduire votre consommation d\'énergie de
                   7%.',
                'impact': 'Moyen',
                'ease': 'Très facile'
            },
            {
                'title': 'Éteignez les appareils en veille',
                'description': 'Les appareils en veille peuvent repré
                   senter jusqu\'à 10% de votre facture d\'électricité.',
                'impact': 'Faible',
                'ease': 'Très facile'
            },
            {
                'title': 'Installez des ampoules LED',
                'description': 'Les ampoules LED consomment jusqu\'à 80% d
                   \'électricité en moins que les ampoules à incandescence
                   .',
                'impact': 'Faible',
                'ease': 'Facile'
            }
        ]
    elif category == 'consumption':
        recommendations = [
            {
                'title': 'Allongez la durée de vie de vos appareils é
                   lectroniques',
                'description': 'La fabrication d\'un smartphone représente
                    environ 80% de son impact environnemental total.',
                'impact': 'Élevé',
                'ease': 'Moyen'
            },
            {
                'title': 'Achetez des vêtements de seconde main',
                'description': 'L\'industrie textile est la 2ème plus
```

```
                    polluante au monde. Privilégiez les vêtements d\'
                    occasion.',
                'impact': 'Moyen',
                'ease': 'Facile'
            },
            {
                'title': 'Réparez au lieu de remplacer',
                'description': 'Réparer un objet plutôt que d\'en acheter
                    un neuf peut réduire significativement votre empreinte
                    carbone.',
                'impact': 'Élevé',
                'ease': 'Moyen'
            }
        ]

        # Retourner les 3 meilleures recommandations pour cette catégorie
        return recommendations

    def _get_generic_recommendations(self):
        """
        Génère des recommandations génériques adaptées à tous les utilisateurs

        Returns:
            list: Liste de recommandations génériques
        """
        return [
            {
                'title': 'Privilégiez les déplacements doux',
                'description': 'Marcher ou faire du vélo pour les courtes
                    distances est bon pour la santé et pour la planète.',
                'impact': 'Moyen',
                'ease': 'Facile'
            },
            {
                'title': 'Réduisez votre consommation de viande',
                'description': 'Essayez d\'introduire une journée sans viande
                    par semaine dans votre alimentation.',
                'impact': 'Élevé',
                'ease': 'Moyen'
            },
            {
                'title': 'Éteignez les lumières inutiles',
                'description': 'Cette simple habitude peut réduire votre
                    consommation d\'électricité jusqu\'à 5%.',
                'impact': 'Faible',
                'ease': 'Très facile'
            },
            {
                'title': 'Limitez les achats en ligne avec livraison express',
                'description': 'Les livraisons express génèrent plus d\'é
                    missions que les livraisons standard qui permettent d\'
                    optimiser les trajets.',
                'impact': 'Moyen',
                'ease': 'Facile'
            },
            {
                'title': 'Utilisez des sacs réutilisables',
                'description': 'Évitez les sacs en plastique à usage unique
                    lors de vos courses.',
                'impact': 'Faible',
                'ease': 'Très facile'
```

```
            }
        ]
```

Listing 5 – Code du module `recommendation.py` (système de recommandations)

Le système de recommandations fonctionne selon la logique suivante :

1. **Analyse des activités récentes** :
   Récupère les 30 derniers jours d'activité de l'utilisateur.

```
        def get_personalized_recommendations(self):
            # Récupérer les activités récentes (30 derniers jours)
            recent_activities = Activity.query.filter(
                Activity.user_id == self.user_id,
                Activity.date >= start_date
            ).all()
```

2. **Fallback intelligent** :
   Si l'utilisateur a moins de 5 activités, des recommandations génériques sont proposées.
3. **Identification de la catégorie principale** :
   Calcule les émissions par catégorie et cible celle qui émet le plus pour des conseils adaptés.
4. **Gestion robuste des erreurs** :
   Garantit le retour de recommandations même en cas d'erreur de données.

```
    try:
        # Logique principale
    except Exception as e:
        # En cas d'erreur, retourner des recommandations génériques
        return self._get_generic_recommendations()
```

5. **Base de connaissances structurée** :
   Pour chaque catégorie (transport, alimentation, énergie, consommation), une base de recommandations détaillées est disponible, incluant titre, description, impact environnemental et facilité de mise en uvre.

3.3 Conclusion

Le module `controllers` occupe une place centrale dans larchitecture dEcoTrace en assurant la logique métier essentielle : calcul précis de lempreinte carbone et génération de recommandations personnalisées. Grâce à une séparation claire des responsabilités, il garantit robustesse, évolutivité et expérience utilisateur optimale, tout en facilitant la maintenance et lajout de nouvelles fonctionnalités.

4 PARTIE III : Le module `auth`

Pour gérer l'authentification des utilisateurs, le module `auth` s'appuie sur `Flask-Login` pour la gestion des sessions. Ce module est organisé de la manière suivante :

- `__init__.py` : Définition du module
- `apps.py` : Configuration du Blueprint Flask pour l'authentification
- `forms.py` : Formulaires de validation pour l'inscription et la connexion
- `models.py` : Modèle de données User avec ses méthodes principales
- `views.py` : Vues basées sur les classes pour gérer les requêtes HTTP

Cette structure permet de séparer clairement la logique métier, la validation des données et la présentation.

```
~/Documents/GitHub/ecotrace-l2/auth git:(main)  5 files changed,
tree

.
├── __init__.py
├── apps.py
├── forms.py
├── models.py
└── views.py

1 directory, 5 files
```

4.1 Configuration Blueprint (`apps.py`)

Ce fichier initialise le module d'authentification en créant un Blueprint Flask et en configurant les routes associées. Voici son contenu :

```python
from flask import Blueprint

auth = Blueprint("auth", __name__)

from .views import (
    RegisterView,
    LoginView,
    LogoutView,
    DashboardView,
)

auth.add_url_rule(
    "/register/",
    view_func=RegisterView.as_view("register"),
)
auth.add_url_rule(
    "/login/",
    view_func=LoginView.as_view("login"),
)
auth.add_url_rule(
    "/logout/",
    view_func=LogoutView.as_view("logout"),
)
auth.add_url_rule(
    "/dashboard/",
    view_func=DashboardView.as_view("dashboard"),
)
```

Listing 6 – Code du module `apps.py` (configuration du Blueprint)

Un Blueprint Flask dédié à l'authentification est créé :

```
auth = Blueprint("auth", __name__)
```

Quatre routes principales sont enregistrées :

- `register` : Inscription des nouveaux utilisateurs
- `login` : Connexion des utilisateurs existants
- `logout` : Déconnexion
- `dashboard` : Tableau de bord personnel

L'utilisation des vues basées sur les classes (`MethodView`) permet une meilleure organisation du code et facilite la maintenance.

4.2 Formulaires de validation (`forms.py`)

Le fichier `forms.py` utilise le module `Flask-WTF` pour la conception et la validation des formulaires. Il définit les formulaires d'inscription et de connexion avec toutes les règles de validation nécessaires. Voici son contenu :

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired, Email, Length, EqualTo,
    ValidationError
from .models import User

class RegistrationForm(FlaskForm):
    name = StringField(
        'Nom',
        validators=[
            DataRequired(message="Le nom est requis"),
            Length(min=2, max=100, message="Le nom doit contenir entre 2 et
                100 caractères")
        ]
    )
    email = StringField(
        'Email',
        validators=[
            DataRequired(message="L'email est requis"),
            Email(message="Veuillez entrer une adresse email valide")
        ]
    )
    password = PasswordField(
        'Mot de passe',
        validators=[
            DataRequired(message="Le mot de passe est requis"),
            Length(min=6, message="Le mot de passe doit contenir au moins 6
                caractères")
        ]
    )
    confirm_password = PasswordField(
        'Confirmer le mot de passe',
        validators=[
            DataRequired(message="La confirmation du mot de passe est requise"
                ),
            EqualTo('password', message="Les mots de passe ne correspondent
                pas")
        ]
    )
```

```
    submit = SubmitField("S'inscrire")

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError("Cette adresse email est déjà utilisée")


class LoginForm(FlaskForm):
    email = StringField('Email', validators=[
        DataRequired(message="L'email est requis"),
        Email(message="Veuillez entrer une adresse email valide")
    ])
    password = PasswordField('Mot de passe', validators=[
        DataRequired(message="Le mot de passe est requis")
    ])
    remember = BooleanField('Se souvenir de moi')
    submit = SubmitField('Se connecter')
```

Listing 7 – Code du module `forms.py` (formulaires d'authentification)

1. **RegistrationForm** :
   Cette classe gère l'inscription des nouveaux utilisateurs. Elle inclut les champs suivants avec validation :

   — Le champ nom (`name`) :

   ```
   name = StringField(
       'Nom',
       validators=[
           DataRequired(message="Le nom est requis"),
           Length(
               min=2,
               max=100,
               message="Le nom doit contenir entre 2 et 100 caractères"
           )
       ]
   )
   ```

   — Le champ adresse mail (`email`) :

   ```
   email = StringField(
       'Email',
       validators=[
           DataRequired(message="L'email est requis"),
           Email(message="Veuillez entrer une adresse email valide")
       ]
   )
   ```

   — Le champ mot de passe (`password`) :

```
password = PasswordField(
    'Mot de passe',
    validators=[
        DataRequired(message="Le mot de passe est requis"),
        Length(
            min=6,
            message="Le mot de passe doit contenir au moins 6 caractères"
        )
    ]
)
```

— Le champ confirmation du mot de passe (`confirm_password`) :

```
confirm_password = PasswordField(
    'Confirmer le mot de passe',
    validators=[
        DataRequired(message="La confirmation du mot de passe est requise"),
        EqualTo(
            'password',
            message="Les mots de passe ne correspondent pas"
        )
    ]
)
```

Pour empêcher la création de comptes avec des emails déjà utilisés, une validation personnalisée est ajoutée dans la méthode `validate_email` :

```
def validate_email(self, email):
    user = User.query.filter_by(email=email.data).first()
    if user:
        raise ValidationError("Cette adresse email est déjà utilisée")
```

2. **LoginForm** :
   Cette classe gère la connexion des utilisateurs existants. Elle inclut les champs suivants :
   (a) Le champ adresse mail (`email`) qui sert à identifier l'utilisateur.
   (b) Le champ mot de passe (`password`)
   (c) La case à cocher "Se souvenir de moi" (`remember`) qui permet de garder l'utilisateur connecté.

4.3 Modèle User (`models.py`)

Le fichier `models.py` définit le modèle de données `User` qui représente les utilisateurs de l'application. Il hérite de `db.Model` de `Flask-SQLAlchemy` et de `UserMixin` pour bénéficier des fonctionnalités de `Flask-Login`. Voici son contenu :

```
from datetime import datetime, timezone
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash

from configs.settings import db
from carbon.models import EmissionFactor


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

```
    """Modèle pour les utilisateurs de l'application"""
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)
    is_admin = db.Column(db.Boolean(), default=False)
    created_at = db.Column(db.DateTime, default=datetime.now(timezone.utc))

    # Relations
    activities = db.relationship('Activity', backref='user', lazy=True)

    def __init__(self, name, email, password):
        self.name = name
        self.email = email
        self.password = generate_password_hash(password)
        self.created_at = datetime.now(timezone.utc)

    def check_password(self, password2):
        """Vérifie le mot de passe de l'utilisateur"""
        return check_password_hash(self.password, password2)

    def get_total_emissions(self):
        """Calcule les émissions totales de l'utilisateur à ce jour"""
        total = 0
        for activity in self.activities:
            factor = EmissionFactor.query.get(activity.emission_factor_id)
            total += activity.quantity * factor.co2_factor

        return total

    @classmethod
    def get_all_users_total_emissions(cls, with_unit=True):
        """
        Retourne les émissions totales de tous les utilisateurs, formatées (ex
            : 1.2 ktCO2, 950 tCO2, 500 kgCO2).
        Cette méthode calcule pour tous les utilisateurs, indépendamment de la
            connexion.
        """
        total = 0
        for user in cls.query.all():
            total += user.get_total_emissions()
        total = round(total, 2)
        if with_unit:
            if total >= 1_000_000:
                return f"{round(total/1_000_000, 2)} MtCO2"
            elif total >= 1_000:
                return f"{round(total/1_000, 2)} ktCO2"
            elif total >= 1:
                return f"{round(total, 2)} tCO2"
            else:
                return f"{round(total*1000, 2)} kgCO2"
        return total

    @classmethod
    def get_total_users(cls, formatted=False):
        """Retourne le nombre total d'utilisateurs, formaté si demandé (ex: 1,
            10, 1K+, 10K+)"""
        count = cls.query.count()
        if formatted:
```

```
            if count >= 10_000:
                return f"{count // 1000}K+"
            elif count >= 1_000:
                return f"{count // 1000}K+"
            else:
                return str(count)
        return count

    @property
    def is_authenticated(self):
        return True

    @property
    def is_active(self):
        return True

    def __repr__(self):
        return f"<User {self.name} ({self.email})>"

    def __str__(self):
        return self.name
```

Listing 8 – Code du module `models.py` (modèle User)

1. **Sécurité des mots de passe** :
   Le module `werkzeug.security` est utilisé pour hacher les mots de passe avant de les stocker dans la base de données :

   ```
   def __init__(self, name, email, password):
       self.password = generate_password_hash(password)
   ```

   Une méthode permet également de vérifier les mots de passe lors de la connexion :

   ```
   def check_password(self, password2):
       return check_password_hash(self.password, password2)
   ```

2. **Relations avec les données carbone** :
   Une relation est établie avec le modèle `Activity` :

   ```
   activities = db.relationship('Activity', backref='user', lazy=True)
   ```

   Cette relation permet d'accéder facilement à toutes les activités d'un utilisateur.
3. **Méthodes de calcul personnalisées** :

   (a) Calcul des émissions personnelles :

   ```
   def get_total_emissions(self):
       total = 0
       for activity in self.activities:
           factor = EmissionFactor.query.get(activity.emission_factor_id)
           total += activity.quantity * factor.co2_factor
       return total
   ```

   (b) Méthodes de classe pour les statistiques globales : Des méthodes de classe fournissent des données à l'ensemble de l'application :

— `get_all_users_total_emissions()` : Calcule et formate les émissions totales de tous les utilisateurs

— `get_total_users()` : Retourne le nombre d'utilisateurs avec formatage optionnel

Le formatage automatique des unités est particulièrement utile :

```
    if total >= 1_000_000:
        return f"{round(total/1_000_000, 2)} MtCO2"
    elif total >= 1_000:
        return f"{round(total/1_000, 2)} ktCO2"
```

4.4 Vues d'authentification (`views.py`)

Le fichier `views.py` contient les vues basées sur les classes pour gérer les requêtes HTTP liées à l'authentification. Voici son contenu :

```
import json
from datetime import datetime

from flask import (
    render_template,
    session,
    redirect,
    url_for,
    flash,
    request
)
from flask.views import MethodView
from flask_login import (
    logout_user,
    login_required,
    login_user,
    current_user
)

from configs.errors import get_form_errors
from controllers.calculator import CarbonCalculator
from controllers.recommendation import RecommendationEngine
from carbon.models import EmissionFactor, Activity

from .forms import (
    RegistrationForm,
    LoginForm
)
from .models import db, User


class RegisterView(MethodView):
    """
    Vue pour l'inscription des utilisateurs.
    Cette vue gère l'affichage du formulaire d'inscription,
    la validation des données soumises et la création d'un nouvel utilisateur.
    """
    template_name = "auth/register.html"
    form_class = RegistrationForm

    def get(self):
        if current_user.is_authenticated:
            # Si l'utilisateur est déjà connecté,
```

```
            # redirigez-le vers le tableau de bord
            flash("Vous êtes déjà connecté.", "info")
            return redirect(url_for('auth.dashboard'))

        form = self.form_class()

        ctx = {
            "title": "Inscription",
            "form": form
        }
        return render_template(self.template_name, **ctx)

    def post(self):
        form = self.form_class()

        if form.validate_on_submit():
            name=form.name.data
            email=form.email.data
            password=form.password.data

            # Vérifier si l'utilisateur existe déjà
            existing_user = User.query.filter_by(email=email).first()

            if existing_user:
                flash("Un utilisateur avec cet email existe déjà.", "error")
                return redirect(url_for('auth.register'))

            new_user = User(
                name=name,
                email=email,
                password=password
            )
            db.session.add(new_user)
            db.session.commit()

            flash("Inscription réussie ! Vous pouvez maintenant vous connecter
                .", "success")
            return redirect(url_for('auth.login'))

        elif form.errors:
            get_form_errors(form)

        ctx = {
            "title": "Inscription",
            "form": form
        }
        return render_template(self.template_name, **ctx)


class LoginView(MethodView):
    """
    Vue pour la connexion des utilisateurs.
    Cette vue gère l'affichage du formulaire de connexion,
    la validation des données soumises et l'authentification de l'utilisateur.
    """
    template_name = "auth/login.html"
    form_class = LoginForm

    def get(self):
        if current_user.is_authenticated:
            # Si l'utilisateur est déjà connecté,
```

```
            # redirigez-le vers le tableau de bord
            flash("Vous êtes déjà connecté.", "info")
            return redirect(url_for('auth.dashboard'))

        form = self.form_class()

        ctx = {
            "title": "Connexion",
            "form": form
        }
        return render_template(self.template_name, **ctx)

    def post(self):
        form = self.form_class()

        if form.validate_on_submit():
            user = User.query.filter_by(email=form.email.data).first()

            if user and user.check_password(form.password.data):
                login_user(
                    user,
                    remember=form.remember.data
                )
                next_page = request.args.get('next')
                flash("Connexion réussie !", "success")
                return redirect(next_page or url_for('auth.dashboard'))
            else:
                flash("Email ou mot de passe incorrect.", "error")

        elif form.errors:
            get_form_errors(form)

        ctx = {
            "title": "Connexion",
            "form": form
        }
        return render_template(self.template_name, **ctx)


class LogoutView(MethodView):
    """
    Vue pour la déconnexion des utilisateurs.
    Cette vue gère la déconnexion de l'utilisateur et le redirige vers la page
        d'accueil ou vers la page de connexion.
    """
    decorators = [login_required]

    def get(self):
        logout_user()
        flash("Déconnexion réussie !", "info")
        session.pop("user_id", None)
        return redirect(url_for("carbon.index")) or redirect(url_for("auth.
            login"))


class DashboardView(MethodView):
    """
    Vue pour le tableau de bord des utilisateurs.
    Cette vue affiche les informations de l'utilisateur connecté,
    y compris les données des émissions de carbone.
    """
```

```
    template_name = "auth/dashboard.html"
    decorators = [login_required]

    def get(self):
        user = current_user
        user_id = user.id if user.is_authenticated else session.get("user_id")

        if not user_id:
            flash("Vous devez être connecté pour accéder au tableau de bord.",
                "warning")
            return redirect(url_for("auth.login"))

        # Créer les calculateurs
        calculator = CarbonCalculator(user_id)
        recommendation_engine = RecommendationEngine(user_id)

        # Obtenir les données avec la gestion des erreurs
        today = datetime.now().date()

        try:
            # Essayer de calculer l'empreinte quotidienne
            daily_footprint = calculator.calculate_daily_footprint(today)

        except Exception as e:
            # En cas d'erreur, utiliser des valeurs par défaut
            daily_footprint = {
                'total': 0,
                'by_category': {
                    'transport': 0,
                    'food': 0,
                    'energy': 0,
                    'consumption': 0
                }
            }

        try:
            # Essayer de calculer la tendance hebdomadaire
            weekly_trend = calculator.calculate_weekly_trend()
            weekly_trend_json = json.dumps(weekly_trend)

        except Exception as e:
            # En cas d'erreur, utiliser des valeurs par défaut
            weekly_trend = []
            weekly_trend_json = "[]"

        try:
            # Essayer de calculer le résumé mensuel
            monthly_summary = calculator.calculate_monthly_summary()
            monthly_summary_json = json.dumps(monthly_summary)
        except Exception as e:
            # En cas d'erreur, utiliser des valeurs par défaut
            monthly_summary = []
            monthly_summary_json = "[]"

        try:
            # Essayer d'obtenir les recommandations
            recommendations = recommendation_engine.
                get_personalized_recommendations()

        except Exception as e:
            # En cas d'erreur, utiliser des valeurs par défaut
```

```
            recommendations = []

        # Convertir les données pour JavaScript
        try:
            categories_json = json.dumps(daily_footprint['by_category'])

        except Exception as e:
            # En cas d'erreur, utiliser une chaîne vide
            categories_json = "{}"

        # Récupérer les 3 dernières activités de l'utilisateur
        activities = Activity.query.filter_by(user_id=user.id).order_by(
            Activity.date.desc()).limit(3).all()

        # Préparer les données pour l'affichage
        activities_data = []

        for activity in activities:
            emission_factor = EmissionFactor.query.get(activity.
                emission_factor_id)
            activities_data.append({
                'id': activity.id,
                'date': activity.date,
                'category': emission_factor.category,
                'name': emission_factor.activity_name,
                'quantity': activity.quantity,
                'unit': emission_factor.unit,
                'emissions': round(activity.quantity * emission_factor.
                    co2_factor, 2)
            })

        ctx = {
            "title": "Tableau de bord",
            "user": current_user,
            "total_emissions": current_user.get_total_emissions(),
            "daily_total": round(daily_footprint.get("total", 0), 2),
            "daily_by_category": daily_footprint.get("by_category", {}),
            "daily_footprint": daily_footprint,

            "weekly_trend": weekly_trend_json,
            "categories_data": categories_json,
            "monthly_summary": monthly_summary_json,

            "recommendations": recommendations,
            "activities": activities_data,
        }
        return render_template(self.template_name, **ctx)
```

Listing 9 – Code du module `views.py` (vues d'authentification)

1. **Inscription (`RegisterView`)** :
   Gère l'inscription des nouveaux utilisateurs à l'aide du formulaire `RegistrationForm`. En cas de succès, l'utilisateur est redirigé vers la page de connexion.
   Cette vue inclut une gestion complète :

   — Gestion des utilisateurs déjà connectés : Redirige les utilisateurs déjà connectés vers le tableau de bord s'ils tentent d'accéder à la page d'inscription.

```
if current_user.is_authenticated:
    flash("Vous êtes déjà connecté.", "info")
    return redirect(url_for('auth.dashboard'))
```

— Vérification de l'unicité de l'email : Vérifie si l'email fourni par l'utilisateur est déjà utilisé. Si c'est le cas, un message d'erreur est affiché.

```
existing_user = User.query.filter_by(email=email).first()
if existing_user:
    flash("Un utilisateur avec cet email existe déjà.", "error")
    return redirect(url_for('auth.register'))
```

2. **Connexion (`LoginView`)** :
Gère la connexion des utilisateurs existants à l'aide du formulaire `LoginForm`. En cas de succès, l'utilisateur est redirigé vers le tableau de bord.
Cette vue inclut :

— Authentification sécurisée : Utilise `Flask-Login` pour vérifier les identifiants et gérer la session utilisateur.

```
        if user and user.check_password(form.password.data):
            login_user(user, remember=form.remember.data)
```

— Redirection intelligente : Utilise la variable `next` pour rediriger l'utilisateur vers la page qu'il souhaitait atteindre avant de se connecter ou vers le tableau de bord par défaut.

```
        next_page = request.args.get('next')
        return redirect(next_page or url_for('auth.dashboard'))
```

3. **Déconnexion (`LogoutView`)** :
Gère la déconnexion des utilisateurs en appelant la méthode `logout_user()` de `Flask-Login`.
4. **Tableau de bord (`DashboardView`)** :
Affiche les statistiques personnelles de l'utilisateur : empreinte carbone quotidienne, hebdomadaire et mensuelle, ainsi que les recommandations personnalisées.
Cette vue intègre :

— Sécurité :

```
        decorators = [login_required]
```

— Gestion robuste des erreurs : Chaque calcul est enveloppé dans des blocs `try/except` pour éviter que l'application ne plante en cas de données corrompues.

```
        try:
            # Code qui pourrait lever une exception
        except Exception as e:
            # Gestion de l'exception
```

— Intégration des contrôleurs : Les classes du module controllers sont utilisées pour fournir des données riches au tableau de bord.

```
calculator = CarbonCalculator(user_id)
recommendation_engine = RecommendationEngine(user_id)
```

— Préparation des données JavaScript : Les données sont préparées au format JSON pour permettre l'affichage de graphiques interactifs côté client.

```
weekly_trend_json = json.dumps(weekly_trend)
categories_json = json.dumps(daily_footprint['by_category'])
```

Le module `auth` fournit ainsi une base d'authentification solide et sécurisée, essentielle pour protéger les données personnelles des utilisateurs d'EcoTrace.

4.5 Conclusion

En conclusion, le module `auth` d'EcoTrace joue un rôle central dans la protection des données des utilisateurs. Grâce à une authentification robuste et à une gestion sécurisée des sessions, seules les personnes autorisées ont accès aux informations sensibles. L'intégration du hachage des mots de passe et de la vérification de l'e-mail renforce encore la sécurité globale de l'application.

5 PARTIE IV : Le module `carbon`

Le module `carbon` gère le coeur fonctionnel de l'application, à savoir le suivi des émissions carbone. Il reprend la même architecture que le module `auth`, assurant ainsi la cohérence et la clarté de l'organisation du projet :

- `__init__.py` : Initialisation du module
- `apps.py` : Configuration du Blueprint pour les fonctionnalités carbone
- `forms.py` : Formulaire pour l'ajout d'activités
- `models.py` : Modèles EmissionFactor et Activity
- `views.py` : Vues pour la gestion des activités carbone

Cette structure garantit une séparation claire des responsabilités et une maintenance facilitée.

```
~/Documents/GitHub/ecotrace-l2/carbon git:(main)  6 files change
tree

.
├── __init__.py
├── apps.py
├── forms.py
├── models.py
└── views.py

1 directory, 5 files
```

5.1 Configuration Blueprint (`apps.py`)

Le Blueprint `carbon` propose quatre routes principales :

```
carbon = Blueprint("carbon", __name__)
```

Les routes définies sont :

- `/` : Page d'accueil de l'application
- `/add_activity/` : Formulaire d'ajout d'activité
- `/history/` : Historique des activités de l'utilisateur
- `/activity/delete/<int:activity_id>/` : Suppression d'une activité spécifique

L'utilisation d'une route paramétrique pour la suppression permet une gestion sécurisée et personnalisée des activités.

5.2 Formulaire d'ajout d'activité `AddActivityForm` (`forms.py`)

Le fichier `forms.py` contient le formulaire d'ajout d'activité, qui permet à l'utilisateur de saisir ses activités quotidiennes. Voici son contenu :

```
from flask_wtf import FlaskForm
from wtforms.fields import (
    StringField,
    DateField,
    SelectField,
    FloatField
)
from wtforms.validators import (
    DataRequired,
    NumberRange,
```

```
    Optional,
)


class AddActivityForm(FlaskForm):
    category = SelectField(
        label="Catégorie",
        choices=[
            ("", "Sélectionnez une catégorie"),
            ("transport", "Transport"),
            ("food", "Alimentation"),
            ("energy", "Énergie"),
            ("consumption", "Consommation"),
        ],
        validators=[DataRequired(message="Le champ catégorie est requis")],
    )
    activity_id = SelectField(
        label="Type d'activité",
        choices=[],  # Les choix seront remplis dynamiquement
        validators=[Optional()],
    )
    quantity = FloatField(
        label="Quantité",
        validators=[
            DataRequired(message="Le champ quantité est requis"),
            NumberRange(min=0, message="La quantité doit être positive"),
        ],
    )
    date = DateField(
        label="Date",
        format="%Y-%m-%d",
        validators=[DataRequired(message="Le champ date est requis")],
    )
    submit = StringField(label="Ajouter l'activité")
```

Listing 10 – Code du module `forms.py` (formulaire d'ajout d'activité)

Ce formulaire comprend les champs suivants :

- `category` : Catégorie de l'activité (transport, alimentation, etc.)
- `activity_id` : Identifiant de l'activité
- `quantity` : Quantité d'émissions (en kgCO2)
- `date` : Date de l'activité

Le formulaire est conçu pour être convivial et intuitif, facilitant la saisie des données par l'utilisateur. Il propose notamment :

1. **Sélection de la catégorie** : L'utilisateur choisit la catégorie de l'activité (transport, alimentation, etc.) via une liste déroulante.

```
category = SelectField(
    label="Catégorie",
    choices=[
        ("", "Sélectionnez une catégorie"),
        ("transport", "Transport"),
        ("food", "Alimentation"),
        ("energy", "Énergie"),
        ("consumption", "Consommation"),
    ],
    validators=[DataRequired(message="Le champ catégorie est requis")],
)
```

2. **Champ d'activité dynamique** : Le champ d'activité s'adapte en fonction de la catégorie sélectionnée. Par exemple, si l'utilisateur choisit "Transport", les options proposées seront spécifiques à cette catégorie.

```
activity_id = SelectField(
    label="Type d'activité",
    choices=[],  // Les choix sont remplis dynamiquement
    validators=[Optional()],
)
```

   Cette approche permet de charger dynamiquement les activités disponibles selon la catégorie sélectionnée.

3. **Validation numérique** : Le champ `quantity` est validé pour garantir une valeur numérique positive, représentant la quantité d'émissions en kgCO2.

```
quantity = FloatField(
    validators=[
        DataRequired(message="Le champ quantité est requis"),
        NumberRange(min=0, message="La quantité doit être positive"),
    ],
)
```

5.3 Modèles de données (`models.py`)

Le fichier `models.py` contient les modèles de données pour les facteurs d'émission et les activités. Voici son contenu :

```python
from datetime import datetime, timezone

from configs.settings import db


class EmissionFactor(db.Model):
    """Modèle pour les facteurs d'émission de CO2"""
    __tablename__ = 'emission_factors'

    id = db.Column(db.Integer, primary_key=True)
    category = db.Column(db.String(50), nullable=False)
    subcategory = db.Column(db.String(50), nullable=False)
    activity_name = db.Column(db.String(100), nullable=False)
    unit = db.Column(db.String(20), nullable=False)
    co2_factor = db.Column(db.Float, nullable=False)
```

```
    source = db.Column(db.String(100))

    # Relations
    activities = db.relationship('Activity', backref='emission_factor', lazy=
        True)

    def __repr__(self):
        return f"<EmissionFactor {self.category}/{self.activity_name}: {self.
            co2_factor} {self.unit}>"

    @classmethod
    def get_by_category(cls, category):
        """Récupère tous les facteurs d'émission d'une catégorie donnée"""
        return cls.query.filter_by(category=category).all()



class Activity(db.Model):
    """Modèle pour les activités générant des émissions de CO2"""
    __tablename__ = 'activities'

    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
    emission_factor_id = db.Column(db.Integer, db.ForeignKey('emission_factors
        .id'), nullable=False)
    quantity = db.Column(db.Float, nullable=False)
    date = db.Column(db.Date, nullable=False, default=datetime.now(timezone.
        utc).date())
    created_at = db.Column(db.DateTime, default=datetime.now(timezone.utc))

    def __repr__(self):
        return f"<Activity {self.id} - User {self.user_id}>"

    def get_emissions(self):
        """Calcule les émissions pour cette activité"""
        factor = EmissionFactor.query.get(self.emission_factor_id)
        return self.quantity * factor.co2_factor

    @classmethod
    def get_total_activities(cls):
        """Retourne le nombre total d'activités, formaté (ex: 1, 1K, 1.2K)"""
        count = cls.query.count()
        if count < 1000:
            return str(count)
        elif count < 1_000_000:
            return f"{count/1000:.1f}K".rstrip('0').rstrip('.')
        else:
            return f"{count/1_000_000:.1f}M".rstrip('0').rstrip('.')
```

Listing 11 – Code du module `models.py` (modèles de données)

1. **Modèle EmissionFactor** :
   Ce modèle représente les facteurs d'émission pour différentes activités. Il comprend les champs suivants :

```
class EmissionFactor(db.Model):
    __tablename__ = 'emission_factors'

    id = db.Column(db.Integer, primary_key=True)
    category = db.Column(db.String(50), nullable=False)
    subcategory = db.Column(db.String(50), nullable=False)
    activity_name = db.Column(db.String(100), nullable=False)
    unit = db.Column(db.String(20), nullable=False)
    co2_factor = db.Column(db.Float, nullable=False)
    source = db.Column(db.String(100))
```

- `id` : Identifiant unique
- `category` : Catégorie de l'activité
- `subcategory` : Sous-catégorie de l'activité
- `activity_name` : Nom de l'activité
- `unit` : Unité de mesure
- `co2_factor` : Facteur d'émission en kgCO2 par unité
- `source` : Source du facteur d'émission (ADEME)

Cette structure permet de constituer une base de données complète avec les données de l'ADEME intégrées dans `settings.py`.
Une méthode de classe utile, `get_by_category`, permet de récupérer les facteurs d'émission par catégorie :

```
@classmethod
def get_by_category(cls, category):
    return cls.query.filter_by(category=category).all()
```

2. **Modèle Activity** :
Ce modèle représente une activité effectuée par l'utilisateur. Il comprend les champs suivants :

```
class Activity(db.Model):
    __tablename__ = 'activities'

    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
    emission_factor_id = db.Column(
        db.Integer,
        db.ForeignKey('emission_factors.id'), nullable=False
    )
    quantity = db.Column(db.Float, nullable=False)
    date = db.Column(
        db.Date,
        nullable=False,
        default=datetime.now(timezone.utc).date()
    )
```

- `id` : Identifiant unique
- `user_id` : Identifiant de l'utilisateur (clé étrangère)
- `emission_factor_id` : Identifiant du facteur d'émission associé
- `quantity` : Quantité d'émissions en kgCO2
- `date` : Date de l'activité

Quelques méthodes utiles :

```
        def get_emissions(self):
            factor = EmissionFactor.query.get(self.emission_factor_id)
            return self.quantity * factor.co2_factor
```

Cette méthode permet de calculer facilement les émissions d'une activité.
Une méthode de classe permet également le formatage dynamique des statistiques :

```
        @classmethod
        def get_total_activities(cls):
            count = cls.query.count()
            if count < 1000:
                return str(count)
            elif count < 1_000_000:
                return f"{count/1000:.1f}K".rstrip('0').rstrip('.')
            else:
                return f"{count/1_000_000:.1f}M".rstrip('0').rstrip('.')
```

5.4 Vues de gestion des activités (`views.py`)

Le fichier `views.py` contient les vues pour gérer les activités carbone. Voici son contenu :

```
from datetime import datetime
from functools import lru_cache
from flask import (
    render_template,
    request,
    redirect,
    url_for,
    flash,
)
from flask.views import MethodView
from flask_login import (
    login_required,
    current_user,
)
from .models import (
    db,
    EmissionFactor,
    Activity
)
from .forms import AddActivityForm


class IndexView(MethodView):
    template_name = "carbon/index.html"

    def get(self):
        ctx = {
            "title": "Accueil"
        }
        return render_template(self.template_name, **ctx)


class AddActivityView(MethodView):
    template_name = "carbon/add_activity.html"
```

```
    form_class = AddActivityForm
    decorators = [login_required]

    VALID_CATEGORIES = {"transport", "food", "energy", "consumption"}

    @lru_cache(maxsize=128)
    def _get_emission_factors(self):
        """Cache des facteurs d'émission pour éviter les requêtes répétées"""
        return {
            category: EmissionFactor.query.filter_by(category=category).all()
            for category in self.VALID_CATEGORIES
        }

    def _get_context(self):
        """Contexte réutilisable pour le template"""
        factors = self._get_emission_factors()
        return {
            "title": "Ajouter une activité",
            "transport_factors": factors["transport"],
            "food_factors": factors["food"],
            "energy_factors": factors["energy"],
            "consumption_factors": factors["consumption"],
            "today": datetime.now().date(),
        }

    def _validate_form_data(self, form_data):
        """Validation centralisée avec retour d'erreurs et de données"""
        errors = []
        validated_data = {}

        # Validation catégorie
        category = form_data.get("category", "").strip()
        if not category:
            errors.append("Veuillez sélectionner une catégorie.")
        elif category not in self.VALID_CATEGORIES:
            errors.append(f"Catégorie '{category}' non valide.")
        else:
            validated_data["category"] = category

        # Validation activité
        activity_id = form_data.get("activity_id", "").strip()
        if not activity_id:
            errors.append("Veuillez sélectionner une activité spécifique.")
        else:
            try:
                activity_id = int(activity_id)
                emission_factor = EmissionFactor.query.get(activity_id)
                if not emission_factor:
                    errors.append(f"L'activité sélectionnée n'existe pas.")
                else:
                    validated_data["emission_factor"] = emission_factor
            except ValueError:
                errors.append("Identifiant d'activité invalide.")

        # Validation quantité
        quantity_str = form_data.get("quantity", "").strip()
        if not quantity_str:
            errors.append("Veuillez saisir une quantité.")
        else:
            try:
                quantity = float(quantity_str)
```

```
                if quantity <= 0:
                    errors.append("La quantité doit être supérieure à zéro.")
                else:
                    validated_data["quantity"] = quantity
            except ValueError:
                errors.append("La quantité n'est pas un nombre valide.")

        # Validation date
        date_str = form_data.get("date", "").strip()
        try:
            date = datetime.strptime(date_str, "%Y-%m-%d").date() if date_str
                else datetime.now().date()
            validated_data["date"] = date
        except ValueError:
            errors.append("Format de date invalide.")

        return errors, validated_data

    def _flash_errors(self, errors):
        """Flash des erreurs de manière groupée"""
        for error in errors:
            flash(error, "danger")

    def _create_activity(self, validated_data):
        """Création sécurisée de l'activité"""
        try:
            activity = Activity(
                user_id=current_user.id,
                emission_factor_id=validated_data["emission_factor"].id,
                quantity=validated_data["quantity"],
                date=validated_data["date"]
            )
            db.session.add(activity)
            db.session.commit()
            return True
        except Exception as e:
            db.session.rollback()
            flash("Une erreur s'est produite lors de l'ajout de l'activité.",
                "danger")
            return False

    def get(self):
        """Affichage du formulaire"""
        return render_template(self.template_name, **self._get_context())

    def post(self):
        """Traitement du formulaire"""
        # Validation des données
        errors, validated_data = self._validate_form_data(request.form)

        if errors:
            self._flash_errors(errors)
            return render_template(self.template_name, **self._get_context())

        # Création de l'activité
        if self._create_activity(validated_data):
            flash("Activité ajoutée avec succès !", "success")
            return redirect(url_for("carbon.add_activity"))

        # En cas d'erreur de création
        return render_template(self.template_name, **self._get_context())
```

```python
class HistoryView(MethodView):
    template_name = "carbon/history.html"
    decorators = [login_required]

    def get(self):
        """Affichage de l'historique des activités"""
        user = current_user

        if not user:
            flash("Vous devez être connecté pour voir votre historique.", "
                warning")
            return redirect(url_for("auth.login"))

        # Récupération des activités de l'utilisateur
        activities = (
            Activity.query
            .filter_by(user_id=current_user.id)
            .order_by(Activity.date.desc())
            .all()
        )

        # Préparation des données pour la contruction de graphiques
        activities_data = []

        for activity in  activities:
            emission_factor = EmissionFactor.query.get(activity.
                emission_factor_id)
            activities_data.append({
                'id': activity.id,
                'date': activity.date,
                'category': emission_factor.category,
                'name': emission_factor.activity_name,
                'quantity': activity.quantity,
                'unit': emission_factor.unit,
                'emissions': round(activity.quantity * emission_factor.
                    co2_factor, 2)
            })
        # Contexte pour le template
        ctx = {
            "title": "Historique des activités",
            "activities": activities,
            "activities_json": activities_data,
        }
        return render_template(self.template_name, **ctx)



class DeleteActivityView(MethodView):
    decorators = [login_required]

    def post(self, activity_id):
        """Suppression d'une activité"""
        user = current_user

        if not user:
            flash("Vous devez être connecté pour supprimer une activité.", "
                warning")
            return redirect(url_for("auth.login"))
```

```
        # Vérification de l'existence de l'activité et appartient # à l'
            utilisateur
        activity = Activity.query.filter_by(
            id=activity_id,
            user_id=user.id
        ).first()

        if not activity:
            flash("Activité non trouvée ou vous n'avez pas la permission de la
                supprimer.", "danger")
            return redirect(url_for("carbon.history"))
        try:
            db.session.delete(activity)
            db.session.commit()
            flash("Activité supprimée avec succès !", "success")

        except Exception as e:
            db.session.rollback()
            flash("Une erreur s'est produite lors de la suppression de l'
                activité.", "danger")

        return redirect(request.referrer) or redirect(url_for("carbon.history"
            ))
```

Listing 12 – Code du module `views.py` (vues de gestion des activités)

1. **Page d'accueil (`IndexView`)** :
   Vue simple pour la page d'accueil, point d'entrée de l'application, affichant les statistiques globales (injectées par les processeurs de contexte).
2. **Ajout d'activité (`AddActivityView`)** :
   Gère le formulaire d'ajout d'activité. En cas de succès, l'utilisateur est redirigé vers la même page pour faciliter l'ajout de plusieurs activités. Cette vue intègre plusieurs innovations :
   - Optimisation avec cache :

     ```
     @lru_cache(maxsize=128)
     def _get_emission_factors(self):
         return {
             category: EmissionFactor.query.filter_by(category=category).all()
             for category in self.VALID_CATEGORIES
         }
     ```

     L'utilisation de `lru_cache` évite de solliciter la base de données à chaque affichage du formulaire.
   - Validation centralisée : Une méthode dédiée valide toutes les données et retourne à la fois les erreurs et les données validées.

```
        def _validate_form_data(self, form_data):
            errors = []
            validated_data = {}

            // Validation catégorie
            category = form_data.get("category", "").strip()
            if not category:
                errors.append("Veuillez sélectionner une catégorie.")
            elif category not in self.VALID_CATEGORIES:
                errors.append(f"Catégorie '{category}' non valide.")
```

— Gestion d'erreurs robuste : Un système de rollback automatique est utilisé en cas d'erreur lors de la création.

```
    def _create_activity(self, validated_data):
        try:
            activity = Activity(...)
            db.session.add(activity)
            db.session.commit()
            return True
        except Exception as e:
            db.session.rollback()
            flash(
                "Une erreur s'est produite lors de l'ajout de l'activité.",
                "danger"
            )
            return False
```

3. **Historique des activités (`ActivityHistoryView`)** :
   Cette vue affiche l'historique des activités de l'utilisateur, offrant un aperçu des actions passées et la possibilité de les gérer.
   Elle inclut :

   — Requête optimisée :

```
        activities = (
            Activity.query
            .filter_by(user_id=current_user.id)
            .order_by(Activity.date.desc())
            .all()
        )
```

   — Préparation des données pour JavaScript : Les données sont formatées pour permettre l'affichage de graphiques côté client.

```
        for activity in activities:
            emission_factor = EmissionFactor.query.get(
                activity.emission_factor_id
            )
            activities_data.append({
                'id': activity.id,
                'date': activity.date,
                'category': emission_factor.category,
                'name': emission_factor.activity_name,
                'quantity': activity.quantity,
                'unit': emission_factor.unit,
                'emissions': round(
                    activity.quantity * emission_factor.co2_factor,
                    2
                )
            })
```

4. **Suppression d'activité (`DeleteActivityView`)** :
   Gère la suppression sécurisée d'une activité spécifique selon son identifiant.

```
activity = Activity.query.filter_by(
    id=activity_id,
    user_id=user.id
).first()

if not activity:
    flash(
        "Activité non trouvée ou vous n'avez pas la permission de la supprimer.",
        "danger"
    )
    return redirect(url_for("carbon.history"))
```

   Cette vérification empêche un utilisateur de supprimer les activités d'autres utilisateurs.
   L'utilisateur est redirigé vers la page précédente, avec un fallback vers l'historique si besoin.

```
    return redirect(request.referrer) or redirect(url_for("carbon.history"))
```

Les vues (`AddActivityView`, `HistoryView`, `DeleteActivityView`) sont protégées par le décorateur `@login_required`, garantissant que seuls les utilisateurs authentifiés peuvent y accéder.
Le module `carbon` constitue le coeur fonctionnel d'EcoTrace et illustre une approche professionnelle du développement web avec Flask.

5.5 Conclusion

En résumé, le module `carbon` d'EcoTrace permet aux utilisateurs de suivre et de gérer leurs émissions de carbone de manière efficace. Grâce à une architecture modulaire et à des vues sécurisées, ils peuvent facilement ajouter, visualiser et supprimer leurs activités tout en bénéficiant d'une expérience utilisateur cohérente et fluide.

6 PARTIE V : Les templates et fichiers statiques

Pour créer une interface utilisateur moderne et cohérente, j'ai utilisé le moteur de rendu Jinja2 et le framework CSS TailwindCSS. Les templates sont organisés selon une architecture hiérarchique :

```
assets/templates/
 auth/                    # Templates d'authentification
    dashboard.html
    login.html
    register.html
 carbon/                  # Templates fonctionnels
    add_activity.html
    history.html
    index.html
 errors/                  # Pages d'erreur
    404.html
    500.html
 partials/               # Composants réutilisables
     base.html
     footer.html
     navbar.html
```

Cette organisation garantit une cohérence visuelle et facilite la maintenance et l'évolution du projet.
Je n'inclus pas ici l'intégralité du code des templates pour des raisons de place ; ils sont disponibles dans le répertoire dédié. Je propose ci-dessous une explication structurée.

LES TEMPLATES

Les templates sont des fichiers HTML qui définissent la structure et le contenu des pages web. Ils utilisent le moteur de templates Jinja2 pour intégrer des données dynamiques et faciliter la réutilisation du code.

```
.
├── auth
│   ├── dashboard.html
│   ├── login.html
│   └── register.html
├── carbon
│   ├── add_activity.html
│   ├── history.html
│   └── index.html
├── errors
│   ├── 404.html
│   └── 500.html
└── partials
    ├── base.html
    ├── footer.html
    └── navbar.html

5 directories, 11 files
```

6.1 Les templates partiels (`partials/`)

```
~/Documents/GitHub/ecotrace-l2/assets/templates/partials git:(main)
tree

.
├── base.html
├── footer.html
└── navbar.html

1 directory, 3 files
```

6.1.1 Template de base (`partials/base.html`)

Le template `base.html` définit la structure HTML commune à toutes les pages, incluant les liens vers les fichiers CSS et JavaScript, ainsi que les blocs pour le contenu dynamique.

1. **Respect des standards web et accessibilité** :

```
<html lang="fr" class="scroll-smooth">
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
```

2. **SEO et métadonnées complètes** :

```
<title>
    {% block title %}
        EcoTrace - Votre compagnon pour un mode de vie durable
    {% endblock %}
</title>
<meta name="description" content="
    {% block description %}
        Mesurez, comprenez et réduisez votre empreinte carbone avec EcoTrace...
    {% endblock %}">
<meta name="keywords" content="empreinte carbone, écologie, développement
durable, EcoTrace, environnement, CO2">
```

   Les blocs `Jinja2` permettent la personnalisation du SEO sur chaque page.

3. **Favicon multi-plateforme** : Un système de favicon exhaustif est mis en place pour tous les appareils et plateformes.

4. **Typographie et design system : Police Inter personnalisée** :

```
@font-face {
    font-family: 'Inter';
    src: url('{{ url_for(
        'static',
        filename='fonts/Inter-VariableFont_opsz,wght.ttf') }}')
        format('truetype-variations');
    font-weight: 100 900;
    font-style: normal;
    font-display: swap;
}
```

   Les fonctionnalités typographiques avancées sont activées pour une meilleure lisibilité :

```
* {
    font-feature-settings: 'cv11', 'ss01';
    font-variant-numeric: tabular-nums;
}
```

5. **Palette de couleurs cohérente** : La palette est basée sur le vert émeraude, en accord avec la thématique environnementale.
6. **Système de messages flash** : Un système de notifications visuelles sophistiqué est mis en place :

```
<div class="flash-message bg-white border-l-4 p-4 rounded-lg
shadow-lg max-w-sm animate-slide-up
{% if category == 'error' %}border-red-500 text-red-700
{% elif category == 'warning' %}border-yellow-500 text-yellow-

{% elif category == 'success' %}border-emerald-500 text-emerald-

{% else %}border-blue-500 text-blue-700{% endif %}">
```

Chaque type de message possède une couleur, une icône, une animation et un positionnement spécifique.

7. **Accessibilité** :
   — Navigation au clavier :

```
<a href="#main-content" class="sr-only focus:not-sr-

focus:top-4 focus:left-4 bg-emerald-600 text-white px-4 py-2
rounded-lg z-50">
    Aller au contenu principal
</a>
```

   — Focus visible :

```
.focus-ring:focus {
    outline: 2px solid #10b981;
    outline-offset: 2px;
}
```

6.1.2 Navigation (`partials/navbar.html`)

Le système de navigation s'adapte automatiquement à l'état de connexion de l'utilisateur.

1. **Navigation desktop** :

```
<div class="hidden lg:flex items-center space-x-2">
    {% if current_user.is_authenticated %}
        <!-- Liens authentifiés -->
    {% else %}
        <!-- Liens publics -->
    {% endif %}
</div>
```

2. **Navigation mobile** :

```
<div class="lg:hidden hidden bg-white border-t border-gray-100
shadow-lg" id="mobile-menu">
```

3. **Logique d'affichage conditionnelle** : La navigation change selon l'état de connexion :
   — Utilisateur connecté : Tableau de bord, Nouvelle activité, Historique, Profil utilisateur, Déconnexion
   — Utilisateur non connecté : Connexion, Inscription
4. **Mise en évidence de la page active** :

```
<a href="{{ url_for('auth.login') }}" class="
{% if request.endpoint == 'auth.login' %}
    bg-gradient-to-r from-emerald-500 to-teal-600 text-white...
{% else %}
    text-gray-700 hover:text-emerald-600...
{% endif %}">
```

6.1.3 Pied de page (`partials/footer.html`)

Le footer est organisé en trois colonnes principales :
— À propos : Logo, description de l'application
— Navigation : Liens contextuels selon l'état d'authentification
— Ressources : FAQ, guides, support (liens prévus pour le développement futur)

1. **Liens dynamiques** : Le footer s'adapte selon que l'utilisateur est connecté ou non.

```
{% if current_user.is_authenticated %}
    <!-- Liens pour utilisateurs connectés -->
{% else %}
    <!-- Liens pour visiteurs -->
{% endif %}
```

2. **Section copyright** :

```
<div class="border-t border-gray-300 bg-white">
    <div class="text-sm text-gray-600">
        © 2025 EcoTrace. Tous droits réservés.
    </div>
```

6.2 Templates d'authentification (`auth/`)

Les templates d'authentification sont situés dans le répertoire `auth/`. Ils incluent les pages de connexion, d'inscription et de tableau de bord utilisateur.

```
~/Documents/GitHub/ecotrace-l2/assets/templates/auth git:(main)  1
tree

.
├── dashboard.html
├── login.html
└── register.html

1 directory, 3 files
```

Tous les templates partagent :

En-tête décoratif commun :

```
<section class="w-full bg-gradient-to-br from-emerald-100 via-teal-50
to-cyan-50 relative overflow-hidden">
    <!-- Formes décoratives en arrière-plan -->
    <div class="absolute inset-0 opacity-15">
        <div class="absolute top-0 left-0 w-80 h-80 bg-gradient-to-r
        from-emerald-300 to-teal-400 rounded-full blur-3xl transform
        -translate-x-1/2 -translate-y-1/2"></div>
        <div class="absolute bottom-0 right-0 w-64 h-64 bg-gradient-to-r
        from-teal-300 to-cyan-400 rounded-full blur-3xl transform
        translate-x-1/2 translate-y-1/2"></div>
    </div>
</section>
```

Cette section crée un arrière-plan dynamique avec des formes floues pour donner de la profondeur visuelle.

6.2.1 Template d'inscription (`auth/register.html`)

1. **Titre et message adaptatif** :

```
<h1 class="text-5xl font-black bg-gradient-to-r from-emerald-700
via-teal-600 to-cyan-600 bg-clip-text text-transparent">
    Rejoignez EcoTrace
</h1>
<p class="text-lg text-gray-700 max-w-2xl leading-relaxed">
    Commencez votre voyage vers un mode de vie plus durable et
    suivez votre empreinte carbone
</p>
```

2. **Indicateurs sociaux** : Des éléments de preuve sociale rassurent les nouveaux utilisateurs.

```
<div class="flex items-center space-x-3 mt-2">
    <div class="flex -space-x-2">
        <div class="w-7 h-7 rounded-full bg-gradient-to-r from-emerald-400
        to-emerald-500 border-2 border-white"></div>
        <div class="w-7 h-7 rounded-full bg-gradient-to-r from-teal-400
        to-teal-500 border-2 border-white"></div>
        <div class="w-7 h-7 rounded-full bg-gradient-to-r from-cyan-400
        to-cyan-500 border-2 border-white"></div>
    </div>
    <span class="text-sm text-gray-600 font-medium">
        Déjà plus de 1000+ utilisateurs engagés
    </span>
</div>
```

3. **Formulaire avec validation visuelle** :

```
<div class="relative">
    <div class="absolute inset-y-0 left-0 pl-3 flex items-center
    pointer-events-none">
        <svg fill="none" stroke="currentColor" class="...">
           ...
        </svg>
    </div>
    {{ form.name(class="block w-full pl-10 pr-3 py-3 border ...") }}
</div>
```

Chaque champ inclut une icône, des bordures arrondies, des états de focus en vert émeraude et une validation d'erreur avec icône.

4. **Gestion d'erreurs intégrée** :

```
 {% if form.name.errors %}
     {% for error in form.name.errors %}
         <p class="text-red-500 text-xs mt-2 flex items-center">
             <svg fill="currentColor" class="w-4 ..." viewBox="0 0 20 20">
                 ...
             </svg>
             {{ error }}
         </p>
     {% endfor %}
 {% endif %}
```

6.2.2 Template de connexion (`auth/login.html`)

Le template de connexion reprend la structure de celui d'inscription, avec des ajustements pour le contexte de connexion.

1. **Titre et message d'accueil** :

```
<h1 class="text-5xl font-black bg-gradient-to-r from-emerald-700 via-teal-

to-cyan-600 bg-clip-text text-transparent">
    Bienvenue sur Ecotrace
</h1>
<p class="text-lg text-gray-700 max-w-2xl leading-relaxed">
    Connectez-vous à votre compte pour continuer votre parcours écologique et
    consulter vos données
</p>
```

2. **Formulaire simplifié** : Le formulaire de connexion est épuré, avec champs email et mot de passe, option "Se souvenir de moi" et lien vers l'inscription.

6.2.3 Template tableau de bord (`auth/dashboard.html`)

Ce template intègre de nombreuses fonctionnalités avancées.

1. **En-tête personnalisé** :

```
<div class="hidden lg:flex flex-col items-end space-y-4">
    <div class="bg-white/70 backdrop-blur-md rounded-2xl p-8 shadow-xl
    border border-white/20">
        <div class="w-16 h-16 bg-gradient-to-r from-emerald-400 to-teal-500
        rounded-2xl flex items-center justify-center shadow-lg mx-auto mb-

            <svg class="w-8 ..." fill="none" stroke="currentColor" ...>
                ...
            </svg>
        </div>
        <div class="text-2xl font-bold ...">
            {{ user.name }}
        </div>
        <div class="text-sm text-gray-600">Heureux de vous revoir</div>
    </div>
</div>
```

2. **Système de cartes statistiques** :

```
<div class="grid grid-cols-1 md:grid-cols-3 gap-6">
    <div class="bg-white rounded-2xl shadow-md border border-gray-100
    p-6 transform transition-all duration-300 hover:-translate-y-1
    hover:shadow-lg">
        <div class="flex items-center justify-between">
            <div>
                <p class="text-sm text-gray-600 ...">
                    Total mensuel
                </p>
                <p class="text-2xl ..." id="totalEmissions">-</p>
                <p class="text-xs text-gray-500">kgCOe</p>
            </div>
            <div class="w-12 h-12 bg-gradient-to-r from-red-500 to-rose-600
            rounded-xl flex items-center justify-center shadow-lg">
                <!-- Icône SVG -->
            </div>
        </div>
    </div>
</div>
```

Les cartes sont animées au survol, avec des icônes colorées et des données mises à jour dynamiquement.

3. **Intégration des graphiques** :

```
<div class="grid grid-cols-1 lg:grid-cols-2 gap-6">
    <div class="bg-white rounded-2xl shadow-md border border-gray-100 p-6">
        <h3 class="text-lg font-bold text-gray-900 mb-4">
            Répartition par catégorie
        </h3>
        <div class="relative h-64">
            <canvas id="categoryChart"></canvas>
        </div>
    </div>
</div>
```

4. **Système de recommandations** :

```
<div class="bg-white rounded-2xl shadow-md border border-gray-100
overflow-hidden">
    <div class="p-6 border-b border-gray-100">
        <div class="flex items-center justify-between">
            <!-- Titre et description -->
            <div class="flex gap-2">
                <button class="px-3 py-2 bg-emerald-100 text-emerald-700
                rounded-lg text-sm font-medium hover:bg-emerald-200
                transition-colors duration-200 filter-btn active"
                data-filter="all">
                    Toutes
                </button>
                <!-- Autres filtres -->
            </div>
        </div>
    </div>
</div>
```

Le système de recommandations propose des filtres interactifs et des badges colorés.

5. **Actions rapides** :

```
<div class="flex flex-col sm:flex-row gap-3">
    <a href="{{ url_for('carbon.add_activity') }}"
    class="bg-gradient-to-r from-emerald-500 to-teal-600 text-white
    px-6 py-3 rounded-xl font-bold hover:from-emerald-600
    hover:to-teal-700 transition-all duration-300 shadow-lg
    hover:shadow-xl transform hover:-translate-y-1 flex
    items-center space-x-2">
        <svg class="w-5 h-5" fill="none" stroke="currentColor" ...>
            ...
        </svg>
        <span>Nouvelle activité</span>
    </a>
</div>
```

6. **Profil utilisateur moderne** :

```
<div class="divide-y divide-gray-100">
    <div class="p-6 hover:bg-gray-50 transition-colors duration-200">
        <div class="flex items-center justify-between">
            <div class="flex items-center space-x-3">
                <div class="w-10 h-10 bg-gradient-to-r from-blue-500
                to-indigo-600 rounded-lg flex items-center justify-center">
                    <svg class="w-5 h-5 text-white" fill="none" ...>
                        ...
                    </svg>
                </div>
                <div>
                    <p class="text-sm font-medium text-gray-900">Nom</p>
                    <p class="text-sm text-gray-500">Votre nom complet</p>
                </div>
            </div>
            <span class="text-sm font-semibold ...">
                {{ user.name }}
            </span>
        </div>
    </div>
</div>
```

7. **Intégration JavaScript** :

```
<script>
    const categoriesData = {{ categories_data| safe }};
    const weeklyData = {{ weekly_trend| safe }};
    const monthlySummary = {{ monthly_summary| safe }};
</script>
<script src="{{ url_for('static', filename='js/dashboard.js') }}"></script>
```

Les données sont préparées côté serveur au format JSON pour Chart.js côté client.

6.3 Templates du module carbone (`carbon/`)

Les templates du module `carbon/` sont conçus pour encourager l'action et l'engagement écologique. Chaque template conserve la cohérence visuelle tout en adaptant le message à sa fonction.

```
~/Documents/GitHub/ecotrace-l2/assets/templates/carbon git:(main)
tree

.
├── add_activity.html
├── history.html
└── index.html

1 directory, 3 files
```

6.3.1 Template d'accueil (`carbon/index.html`)

1. **Page d'accueil marketing complète** :

```
<h1 class="text-5xl font-black bg-gradient-to-r from-emerald-700
via-teal-600 to-cyan-600 bg-clip-text text-transparent">
    Bienvenue sur EcoTrace
</h1>
<p class="text-xl text-gray-700 max-w-2xl leading-relaxed">
    Transformez votre mode de vie avec l'application qui vous aide à
    mesurer, comprendre et réduire votre empreinte carbone au quotidien.
</p>
```

2. **Intégration de données dynamiques** :

```
<span class="text-sm text-gray-600 font-medium">
    Rejoignez plus de {{ total_users }} éco-citoyens engagés
</span>
```

Les variables de contexte affichent des statistiques en temps réel pour renforcer la crédibilité et l'engagement communautaire.

3. **Boutons d'appel à l'action** :

```
<div class="flex flex-col sm:flex-row gap-4 mt-8">
    <a href="{{ url_for('auth.register') }}"
    class="bg-gradient-to-r from-emerald-500 to-teal-600 text-white
     px-8 py-4 rounded-xl font-bold hover:from-emerald-600
     hover:to-teal-700 transition-all duration-300 shadow-lg
     hover:shadow-xl transform hover:-translate-y-1 flex items-center
     justify-center space-x-2">
        <svg fill="none" stroke="currentColor" class="w-5 h-5" ...>
            ...
        </svg>
        <span>Commencer maintenant</span>
    </a>
</div>
```

6.3.2 Template d'ajout d'activité (`carbon/add_activity.html`)

Ce template intègre de nombreuses innovations UX.

1. **Sélecteur de catégories interactif** :

```
<div class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-4">
    <div class="category-option cursor-pointer group"
    data-category="transport">
        <input type="radio" name="category" id="transport" value="transport"
         class="hidden"/>
        <label for="transport" class="block border-2 border-gray-200
        rounded-xl p-6 text-center hover:border-emerald-400
        hover:bg-emerald-50 transition-all duration-300
        group-hover:shadow-lg group-hover:scale-105">
            <div class="w-12 h-12 bg-blue-100 rounded-xl flex
            items-center justify-center group-hover:bg-blue-500
            transition-colors duration-300">
                <svg fill="none" stroke="currentColor" ...>
                    ...
                </svg>
            </div>
        </label>
    </div>
</div>
```

Chaque catégorie possède une couleur thématique, des animations au survol et des icônes SVG personnalisées.

2. **Dropdowns personnalisés avancés** :

```
<div class="custom-dropdown relative" data-target="transport">
    <div class="selected-option flex justify-between items-center
    w-full px-4 py-3 border-2 border-gray-200 placeholder-gray-500
    text-gray-900 rounded-xl focus:outline-none focus:ring-2
    focus:ring-emerald-500 focus:border-emerald-500 bg-white
    cursor-pointer hover:border-emerald-300 transition-all
    duration-300 shadow-sm">
        <span class="selected-text text-gray-500">
            Sélectionnez un type de transport
        </span>
        <svg fill="none" stroke="currentColor" ...>
            ...
        </svg>
    </div>
    <div class="options-container hidden absolute z-10 left-0
    right-0 mt-2 max-h-60 overflow-y-auto bg-white border
    border-gray-200 rounded-xl shadow-xl">
        {% for factor in transport_factors %}
        <div class="option px-4 py-3 hover:bg-emerald-50
        cursor-pointer border-b border-gray-100 text-sm
        transition-colors duration-200"
        data-value="{{ factor.id }}" data-unit="{{ factor.unit }}">
            {{ factor.activity_name }}
        </div>
        {% endfor %}
    </div>
</div>
```

Les dropdowns sont entièrement personnalisés pour une meilleure expérience utilisateur.

3. **Calendrier personnalisé** :

```
<div class="custom-date-dropdown relative overflow-visible">
    <div class="calendar-container hidden absolute z-50 left-0
    right-0 mt-2 bg-white border border-gray-200 rounded-xl
    shadow-xl p-4">
        <div class="calendar-header flex justify-between
        items-center mb-4">
            <button type="button" class="prev-month p-2
            hover:bg-emerald-50 rounded-full transition-colors
            duration-200">
                <svg fill="none" stroke="currentColor" ...>
                    ...
                </svg>
            </button>
            <div class="month-year font-semibold text-gray-900"></div>
            <button type="button" class="next-month p-2 hover:bg-emerald-50
            rounded-full transition-colors duration-200">
                <svg fill="none" stroke="currentColor" ...>
                    ...
                </svg>
            </button>
        </div>
        <div class="weekdays grid grid-cols-7 gap-1 mb-2 text-center
        text-xs text-gray-500 font-medium">
            <div>Lu</div>
            <div>Ma</div>
            <div>Me</div>
            <div>Je</div>
            <div>Ve</div>
            <div>Sa</div>
            <div>Di</div>
        </div>
        <div class="days-grid grid grid-cols-7 gap-1"></div>
    </div>
</div>
```

Le calendrier permet la navigation entre les mois et une sélection rapide de la date.

6.3.3 Template d'historique (`carbon/history.html`)

Ce template affiche l'historique des activités de l'utilisateur avec une interface lisible et intuitive.

1. **Affichage conditionnel intelligent** :

```
{% if activities_json and activities_json|length > 0 %}
    <!-- Contenu avec données -->
{% else %}
    <!-- État vide avec encouragement -->
    <div class="bg-white rounded-2xl shadow-md border
    border-gray-100 overflow-hidden">
        <div class="text-center py-16 px-8">
            <div class="w-24 h-24 bg-gradient-to-r
            from-emerald-400 to-teal-500 rounded-full flex
            items-center justify-center shadow-lg mx-auto mb-8">
                <svg fill="none" stroke="currentColor" ...>
                    ...
                </svg>
            </div>
            <h2 class="text-2xl font-bold text-gray-900 mb-4">
                Votre historique est encore vide
            </h2>
        </div>
    </div>
{% endif %}
```

2. **Statistiques calculées dynamiquement** :

```
<div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
    <div class="bg-white rounded-2xl shadow-md border border-gray-100
    p-6 transform transition-all duration-300 hover:-translate-y-1
    hover:shadow-lg">
        <div class="flex items-center justify-between">
            <div>
                <p class="text-sm text-gray-600 font-medium">
                    Total activités
                </p>
                <p class="text-2xl font-bold text-gray-900">
                    {{ activities_json|length }}
                </p>
                <p class="text-xs text-gray-500">enregistrées</p>
            </div>
            <div class="w-12 h-12 bg-gradient-to-r from-emerald-500
            to-teal-600 rounded-xl flex items-center justify-center
            shadow-lg">
                <svg fill="none" stroke="currentColor" ...">
                    ...
                </svg>
            </div>
        </div>
    </div>
</div>
```

3. **Interface responsive avancée** :

```
<!-- Version desktop : tableau -->
<div class="hidden lg:block overflow-x-auto">
    <table class="min-w-full divide-y divide-gray-200" id="activitiesTable">
        <thead class="bg-gray-50">
            <tr>
                <th class="px-6 py-3 text-left text-xs font-medium
                text-gray-500 uppercase tracking-wider">Date</th>
                <!-- Autres colonnes -->
            </tr>
        </thead>
    </table>
</div>

<!-- Version mobile : cartes -->
<div class="lg:hidden p-6 space-y-4" id="activitiesCards">
    {% for activity in activities_json %}
    <div class="bg-gray-50 rounded-xl p-4 activity-card">
        <!-- Contenu de la carte -->
    </div>
    {% endfor %}
</div>
```

Deux interfaces sont proposées : tableau pour desktop, cartes pour mobile.

4. **Badges de catégories colorés** :

```
{% if activity.category == 'transport' %}
<span class="inline-flex items-center px-2.5 py-0.5 rounded-full
text-xs font-medium bg-blue-100 text-blue-800">
    <svg fill="none" stroke="currentColor" ...">
        ...
    </svg>
    Transport
</span>
{% elif activity.category == 'food' %}
<span class="inline-flex items-center px-2.5 py-0.5 rounded-full
text-xs font-medium bg-orange-100 text-orange-800">
    <svg fill="none" stroke="currentColor" ...">
        ...
    </svg>
    Alimentation
</span>
{% endif %}
```

6.4 Templates d'erreur (`errors/`)

Les pages d'erreur sont conçues pour transformer une expérience négative en opportunité d'engagement, en restant cohérentes avec l'identité écologique d'EcoTrace.

```
~/Documents/GitHub/ecotrace-l2/assets/templates/errors git:(main)
tree

.
├── 404.html
└── 500.html

1 directory, 2 files
```

6.4.1 Template 404 (`errors/404.html`)

1. **Message humoristique et thématique** :

```
<h1 class="text-5xl font-black bg-gradient-to-r from-emerald-700
via-teal-600 to-cyan-600 bg-clip-text text-transparent">
    Page introuvable
</h1>
<p class="text-xl text-gray-700 max-w-2xl leading-relaxed">
    Oups ! Il semblerait que cette page ait pris un chemin plus
    écologique et ait disparu de nos serveurs.
</p>
```

2. **Indicateurs visuels créatifs** :

```
<div class="flex -space-x-2">
    <div class="w-8 h-8 rounded-full bg-gradient-to-r from-red-400
    to-red-500 border-2 border-white flex items-center justify-center">
        <span class="text-xs text-white font-bold">!</span>
    </div>
    <div class="w-8 h-8 rounded-full bg-gradient-to-r from-yellow-400
    to-yellow-500 border-2 border-white flex items-center justify-center">
        <span class="text-xs text-white font-bold">?</span>
    </div>
    <div class="w-8 h-8 rounded-full bg-gradient-to-r from-emerald-400
    to-emerald-500 border-2 border-white flex items-center justify-center">
        <span class="text-xs text-white font-bold"></span>
    </div>
</div>
```

3. **Section principale engageante** :

```
<h2 class="text-3xl font-bold text-gray-900 mb-6">
    Ne perdons pas notre chemin vers la durabilité !
</h2>
<p class="text-lg text-gray-600 mb-8 leading-relaxed max-w-2xl mx-auto">
    Cette page semble avoir migré vers un serveur plus écologique.
    Pendant que nous la retrouvons, explorons d'autres façons de
    réduire votre empreinte carbone.
</p>
```

4. **Navigation contextuelle intelligente** :

```
{% if current_user.is_authenticated %}
    <a href="{{ url_for('auth.dashboard') }}"
    class="bg-white border-2 border-emerald-300 text-emerald-700
    px-8 py-4 rounded-xl font-bold hover:bg-emerald-50
    hover:border-emerald-400 transition-all duration-300 shadow-md
    hover:shadow-lg flex items-center justify-center space-x-2">
        <span>Mon tableau de bord</span>
    </a>
{% else %}
    <a href="{{ url_for('auth.register') }}" class="bg-white
    border-2 border-emerald-300 text-emerald-700 px-8 py-4
    rounded-xl font-bold hover:bg-emerald-50
    hover:border-emerald-400 transition-all duration-300
    shadow-md hover:shadow-lg flex items-center justify-center
    space-x-2">
        <span>Commencer avec EcoTrace</span>
    </a>
{% endif %}
```

5. **Pages populaires adaptatives** :

```
<div class="grid grid-cols-1 md:grid-cols-3 gap-8 max-w-5xl mx-auto">
    <a href="{{ url_for('carbon.index') }}" class="group bg-white
    rounded-2xl shadow-md border border-gray-100 p-8 hover:shadow-lg
    hover:-translate-y-1 transition-all duration-300">
        <div class="w-16 h-16 bg-gradient-to-r from-emerald-500
        to-teal-600 rounded-2xl flex items-center justify-center
        shadow-lg mb-6 group-hover:scale-105 transition-transform
        duration-300">
            <svg fill="none" stroke="currentColor" ...>
                ...
            </svg>
        </div>
        <h4 class="text-xl font-bold text-gray-900 mb-3">Accueil</h4>
        <p class="text-gray-600 leading-relaxed">
            Découvrez EcoTrace et commencez votre parcours vers un
            mode de vie durable.
        </p>
    </a>
</div>
```

6.4.2 Template 500 (`errors/500.html`)

1. **Thématique de maintenance écologique** :

```
<h1 class="text-5xl font-black bg-gradient-to-r from-emerald-700
via-teal-600 to-cyan-600 bg-clip-text text-transparent">
    Serveur en maintenance
</h1>
<p class="text-xl text-gray-700 max-w-2xl leading-relaxed">
    Notre serveur prend une pause écologique pour se régénérer.
    Nous travaillons activement pour rétablir le service.
</p>
```

2. **Indicateurs progressifs** :

```
<div class="flex -space-x-2">
    <div class="w-8 h-8 rounded-full bg-gradient-to-r from-red-400
    to-red-500 border-2 border-white flex items-center justify-center">
        <svg fill="none" stroke="currentColor" ...>
            ...
        </svg>
    </div>
    <div class="w-8 h-8 rounded-full bg-gradient-to-r from-orange-400
    to-orange-500 border-2 border-white flex items-center justify-center">
        <svg fill="none" stroke="currentColor" ...>
            ...
        </svg>
    </div>
    <div class="w-8 h-8 rounded-full bg-gradient-to-r from-emerald-400
    to-emerald-500 border-2 border-white flex items-center justify-center">
        <svg fill="none" stroke="currentColor" ...>
            ...
        </svg>
    </div>
</div>
```

3. **Guide d'action utilisateur** :

```
<div class="grid grid-cols-1 md:grid-cols-3 gap-6">
    <div class="text-center p-6 bg-gradient-to-r from-blue-50 to-blue-100
    rounded-xl border border-blue-200">
        <div class="w-12 h-12 bg-blue-500 rounded-xl flex items-center
        justify-center mx-auto mb-4">
            <svg fill="none" stroke="currentColor"...>
                ...
            </svg>
        </div>
        <h4 class="font-semibold text-gray-900 mb-2">
            Rechargez la page
        </h4>
        <p class="text-sm text-gray-600">
            Actualisez votre navigateur dans quelques minutes
        </p>
    </div>
</div>
```

4. **Bouton de rechargement interactif** :

```
<button onclick="window.location.reload()"
class="bg-gradient-to-r from-emerald-500 to-teal-600 text-white
px-8 py-4 rounded-xl font-bold hover:from-emerald-600
hover:to-teal-700 transition-all duration-300 shadow-lg
hover:shadow-xl transform hover:-translate-y-1 flex items-center
justify-center space-x-2">
    <svg fill="none" stroke="currentColor" ...>
        ...
    </svg>
    <span>Réessayer maintenant</span>
</button>
```

5. **Section engagement environnemental** :

```
<div class="bg-gradient-to-r from-gray-50 to-gray-100 rounded-2xl
p-8 border border-gray-200">
    <h3 class="text-xl font-bold text-gray-900 mb-6 text-center">
        Notre engagement environnemental
    </h3>
    <div class="grid grid-cols-1 md:grid-cols-2 gap-6">
        <div class="flex items-start space-x-3">
            <div class="w-8 h-8 bg-emerald-500 rounded-lg flex
            items-center justify-center flex-shrink-0">
                <svg fill="none" stroke="currentColor"...>...</svg>
            </div>
            <div>
                <h4 class="font-semibold text-gray-900 mb-1">
                    Serveurs verts
                </h4>
                <p class="text-sm text-gray-600">
                    Alimentés par des énergies renouvelables
                </p>
            </div>
        </div>
    </div>
</div>
```

LES STATIQUES

Les fichiers statiques (CSS, JavaScript, fonts, images, favicon) sont gérés par Flask et organisés dans le répertoire `static/`. Ils sont utilisés pour styliser les templates et ajouter des fonctionnalités interactives.

6.5 Les fichiers JavaScript (`static/js`)

J'ai organisé le code JavaScript en modules spécialisés selon leurs responsabilités :

```
~/Documents/GitHub/ecotrace-l2/assets/static/js git:(main)  10 fil
tree

.
├── add_activity.js
├── chart.min.js
├── dashboard.js
├── history.js
└── main.js
```

— `main.js` : Fonctionnalités globales et utilitaires partagés

— `add_activity.js` : Logique spécifique au formulaire d'ajout d'activité

— `dashboard.js` : Gestion du tableau de bord et des graphiques

— `history.js` : Interface d'historique avec graphiques et tri

— `chart.min.js` : Bibliothèque Chart.js pour les visualisations

Cette approche modulaire facilite la maintenance et optimise les performances en chargeant uniquement les scripts nécessaires à chaque page.

6.5.1 Fichier principal (`main.js`)

— **Architecture globale et utilitaires** :

```
document.addEventListener('DOMContentLoaded', function() {
    initMobileNavigation();
    initFlashMessages();
    initSmoothScrolling();
    initLazyLoading();
});
```

— **Gestion du loader de page** :

```
window.addEventListener('load', function() {
    const loader = document.getElementById('page-loader');
    if (loader) {
        loader.style.opacity = '0';
        setTimeout(() => {
            loader.style.display = 'none';
        }, 500);
    }
});
```

— **Navigation mobile avancée**

```
function initMobileNavigation() {
    function toggleMobileMenu() {
        const isHidden = mobileMenu.classList.contains('hidden');

        if (isHidden) {
            mobileMenu.classList.remove('hidden');
            menuIcon.classList.add('hidden');
            closeIcon.classList.remove('hidden');
            document.body.style.overflow = 'hidden'; // Empêcher le scroll
        } else {
            closeMobileMenu();
        }
    }
}
```

Fonctionnalités avancées :

- Prévention du scroll arrière pendant l'ouverture du menu
- Fermeture automatique lors des clics externes
- Fermeture avec la touche Échap
- Fermeture automatique lors du redimensionnement vers desktop
- Fermeture lors des clics sur les liens de navigation

— **Système de messages flash intelligent** :

```
function initFlashMessages() {
    flashMessages.forEach(message => {
        // Créer le bouton de fermeture
        const closeButton = document.createElement('button');
        closeButton.addEventListener('click', () => removeFlashMessage(message));

        // Auto-hide après 5 secondes
        setTimeout(() => {
            removeFlashMessage(message);
        }, 5000);
    });
}
```

— **Animation de suppression** :

```
function removeFlashMessage(message) {
    if (message && message.parentNode) {
        message.style.opacity = '0';
        message.style.transform = 'translateX(100%)';
        setTimeout(() => {
            if (message.parentNode) {
                message.remove();
            }
        }, 300);
    }
}
```

— **Utilitaires et optimisations** :

— Bouton retour en haut dynamique :

```
        window.addEventListener('scroll', function() {
            const backToTop = document.getElementById('back-to-top');
            if (backToTop) {
                if (window.pageYOffset > 300) {
                    backToTop.style.opacity = '1';
                    backToTop.style.pointerEvents = 'all';
                } else {
                    backToTop.style.opacity = '0';
                    backToTop.style.pointerEvents = 'none';
                }
            }
        });
```

— Lazy loading avec IntersectionObserver :

```
    function initLazyLoading() {
        if ('IntersectionObserver' in window) {
            const imageObserver = new IntersectionObserver(
                (entries, observer) => {
                entries.forEach(entry => {
                    if (entry.isIntersecting) {
                        const img = entry.target;
                        if (img.dataset.src) {
                            img.src = img.dataset.src;
                            img.classList.remove('lazy');
                            imageObserver.unobserve(img);
                        }
                    }
                });
            }, {
                rootMargin: '50px 0px',
                threshold: 0.01
            });
        }
    }
```

6.5.2 Formulaire d'ajout d'activité (`add_activity.js`)

Ce script gère la logique du formulaire d'ajout d'activité, avec des fonctionnalités avancées pour améliorer l'expérience utilisateur.

1. **Architecture moderne et optimisée** : J'ai développé une approche moderne avec des sélecteurs optimisés :

```
const $ = (s) => document.querySelector(s);
const $ = (s) => document.querySelectorAll(s);

const state = {
    activeCategory: null,
    currentMonth: new Date().getMonth(),
    currentYear: new Date().getFullYear(),
    selectedDate: new Date($("#date").value || new Date()),
    today: new Date(),
};
```

2. **Gestion des dropdowns personnalisés** :
   — Système de fermeture globale :

```
const closeAll = () => {
    els.dropdowns.forEach((d) => {
        add(d.querySelector(".options-container"), "hidden");
        const chevron = d.querySelector(".chevron-icon");
        if (chevron) chevron.style.transform = "rotate(0deg)";
    });
};
```

   — Délégation d'événements optimisée :

```
document.addEventListener("click", (e) => {
    if (!e.target.closest(".custom-dropdown, .custom-date-

        closeAll();

    // Dropdown toggle
    const selectedOption = e.target.closest(".selected-option");
    if (selectedOption) {
        e.stopPropagation();
        const dropdown = selectedOption.closest(".custom-dropdown");
        const container = dropdown.querySelector(".options-

        const chevron = dropdown.querySelector(".chevron-icon");
        const isOpen = !hasClass(container, "hidden");

        closeAll();
        if (!isOpen) {
            remove(container, "hidden");
            if (chevron) chevron.style.transform = "rotate(180deg)";
        }
    }
});
```

3. **Calendrier personnalisé complet** :

```
const generateCalendar = () => {
    cal.grid.innerHTML = "";
    const monthNames = ["Janvier", "Février", "Mars", ...];
    cal.monthYear.textContent = `${monthNames[state.currentMonth]}
                                ${state.currentYear}`;

    const firstDay = new Date(state.currentYear, state.currentMonth, 1).getDay();
    const adjustedFirstDay = firstDay === 0 ? 6 : firstDay - 1;
    const daysInMonth = new Date(state.currentYear, state.currentMonth + 1, 0)
                        .getDate();

    // Empty cells + days
    for (let i = 0; i < adjustedFirstDay + daysInMonth; i++) {
        const cell = document.createElement("div");
        if (i < adjustedFirstDay) {
            cell.className = "text-center py-2";
        } else {
            const day = i - adjustedFirstDay + 1;
            const currentDate = new Date(
                state.currentYear,
                state.currentMonth,
                day
            );
            const isFuture = currentDate > state.today;
            const isSelected = currentDate.toDateString()
                               === state.selectedDate.toDateString();
            const isToday = currentDate.toDateString()
                            === state.today.toDateString();

            cell.className = `text-center py-2 cursor-pointer rounded-lg
            text-sm font-medium transition-all duration-200 ${
                isSelected ? "bg-emerald-600 text-white shadow-lg" :
                isToday ? "bg-emerald-100 text-emerald-800 ring-2
                ring-emerald-200" :
                isFuture ? "text-gray-300 cursor-not-allowed" :
                "hover:bg-emerald-50 hover:text-emerald-700"
            }`;
        }
    }
};
```

4. **Positionnement dynamique du calendrier** :

```
setTimeout(() => {
    const rect = cal.container.getBoundingClientRect();
    const viewportHeight = window.innerHeight;

    if (rect.bottom > viewportHeight - 20) {
        // Positionner le calendrier au-dessus du champ
        cal.container.style.bottom = "100%";
        cal.container.style.top = "auto";
        cal.container.style.marginBottom = "8px";
        cal.container.style.marginTop = "0";
    }
}, 10);
```

5. **Validation côté client** :

```
els.form.onsubmit = (e) => {
    const errors = [
        {
            test: !$('input[name="category"]:checked'),
            msg: "Veuillez sélectionner une catégorie.",
        },
        {
            test: !els.selectedActivityInput.value,
            msg: "Veuillez sélectionner une activité spécifique.",
        },
        {
            test: !$("#quantity").value || parseFloat($("#quantity").value) <= 0,
            msg: "Veuillez saisir une quantité valide (supérieure à zéro).",
        },
        {
            test: !$("#date").value,
            msg: "Veuillez sélectionner une date.",
        },
    ]
    .filter((v) => v.test)
    .map((v) => v.msg);

    if (errors.length) {
        e.preventDefault();
        $(".error-message").forEach((el) => el.remove());

        errors.forEach((msg) => {
            const div = document.createElement("div");
            div.className = "error-message w-full relative flex
            items-center justify-between p-4 mb-4 rounded-xl
            border border-solid border-red-300 text-red-800
            bg-red-50 shadow-sm";
            div.innerHTML = `...`;
            els.form.parentNode.insertBefore(div, els.form);
            setTimeout(() => div.remove(), 5000);
        });
    }
};
```

6.5.3 Tableau de bord (`dashboard.js`)

1. **Animations et statistiques** :

```
document.addEventListener('DOMContentLoaded', function () {
    updateStatistics();
    createCharts();

    // Animation des cartes statistiques
    const stats = document.querySelectorAll('[class*="text-2xl font-bold"]');
    stats.forEach((stat, index) => {
        stat.style.opacity = '0';
        stat.style.transform = 'translateY(20px)';
        setTimeout(() => {
            stat.style.transition = 'all 0.6s ease-out';
            stat.style.opacity = '1';
            stat.style.transform = 'translateY(0)';
        }, index * 100);
    });
});
```

2. **Intégration Chart.js** :
   — Graphique en doughnut pour les catégories :

```
function createCurrentCategoryChart() {
    const categoryColors = {
        transport: '#3B82F6', // blue-500
        food: '#F97316', // orange-500
        energy: '#EF4444', // red-500
        consumption: '#8B5CF6' // purple-500
    };

    new Chart(ctx, {
        type: 'doughnut',
        data: {
            labels: Object.keys(categoriesData).map(cat =>
            categoryLabels[cat] || cat),
            datasets: [{
                data: Object.values(categoriesData),
                backgroundColor: Object.keys(categoriesData).map(cat =>
                categoryColors[cat] || '#6B7280'),
                borderWidth: 2,
                borderColor: '#ffffff'
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            plugins: {
                tooltip: {
                    callbacks: {
                        label: function (context) {
                            const total = context.dataset.data.reduce(
                                (a, b) => a + b, 0);
                            const percentage = (
                                (context.parsed / total) * 100).toFixed(1);
                            return `${context.label}: ${
                                context.parsed.toFixed(2)}
                                kgCOe (${percentage}%)`;
                        }
                    }
                }
            }
        }
    });
}
```

3. **Système de filtrage des recommandations** :

```
document.addEventListener('DOMContentLoaded', function () {
    const filterButtons = document.querySelectorAll('.filter-btn');
    const recommendationCards = document.querySelectorAll('.recommendation-

    const noRecommendationsMessage = document.getElementById('no-


    filterButtons.forEach(button => {
        button.addEventListener('click', function () {
            const filter = this.getAttribute('data-filter');

            // Update active button
            filterButtons.forEach(btn => {
                btn.classList.remove('active', 'bg-emerald-100', 'text-

                btn.classList.add('bg-gray-100', 'text-gray-600');
            });

            // Filter recommendations
            let visibleCount = 0;

            recommendationCards.forEach(card => {
                const impact = card.getAttribute('data-impact');
                const ease = card.getAttribute('data-ease');
                let shouldShow = false;

                switch (filter) {
                    case 'all': shouldShow = true; break;
                    case 'high-impact': shouldShow = impact === 'Élevé'; break;
                    case 'easy': shouldShow = ease === 'Facile'; break;
                }

                if (shouldShow) {
                    card.style.display = 'block';
                    visibleCount++;
                } else {
                    card.style.display = 'none';
                }
            });

            // Show/hide no results message
            if (visibleCount === 0) {
                noRecommendationsMessage.classList.remove('hidden');
            } else {
                noRecommendationsMessage.classList.add('hidden');
            }
        });
    });
});
```

6.5.4 Historique (`history.js`)

1. **Calculs et visualisations avancées** :

```
function calculateHistoryStats() {
    // Calcul du total des émissions
    const totalEmissions = activitiesData.reduce(
        (sum, activity) => sum + parseFloat(activity.emissions), 0);

    // Calcul des dates de première et dernière activité
    const sortedDates = activitiesData
        .map(activity => new Date(activity.date))
        .sort((a, b) => a - b);

    if (sortedDates.length > 0) {
        const firstDate = sortedDates[0];
        const lastDate = sortedDates[sortedDates.length - 1];

        firstElement.textContent = firstDate.toLocaleDateString(
            'fr-FR', {
            day: 'numeric',
            month: 'short'
        });
    }
}
```

2. **Graphique de tendance historique** :

```
function createHistoryChart() {
    // Regroupement des données par jour
    const dailyEmissions = {};
    activitiesData.forEach(activity => {
        const date = new Date(activity.date);
        const dateKey = date.toISOString().split('T')[0];

        if (!dailyEmissions[dateKey]) {
            dailyEmissions[dateKey] = 0;
        }
        dailyEmissions[dateKey] += parseFloat(activity.emissions);
    });

    // Préparation des données pour Chart.js
    const sortedDates = Object.keys(dailyEmissions).sort();
    const labels = sortedDates.map(dateStr => {
        const date = new Date(dateStr);
        return date.toLocaleDateString('fr-FR', {
            day: 'numeric',
            month: 'short'
        });
    });
}
```

6.5.5 La librairie de graphiques (`chart.js`)

Pour les graphiques, j'ai utilisé la librairie Chart.js, qui est légère et facile à intégrer. Disponible ici : https://www.chartjs.org/

6.6 Le fichier de style (`css/style.css`)

Ce fichier contient les styles globaux de lapplication, générés lors du build avec Tailwind CSS. Il définit lapparence de linterface utilisateur : mise en page, couleurs, polices et composants. Lutilisation de TailwindCSS garantit un design moderne, réactif et cohérent grâce à une approche utilitaire.

```
~/Documents/GitHub/ecotrace-l2/assets/static/css git:(main)  10 fil
tree

.
└── style.css

1 directory, 1 file
```

6.7 La fonte personnalisée Inter (`static/fonts/`)

J'ai intégré la fonte Inter, une police moderne et lisible, pour améliorer l'expérience utilisateur. Elle est chargée via le fichier CSS et utilisée dans toute l'application.

6.8 Les favicons (`static/favicon/`)

Les favicons sont des petites icônes qui représentent l'application dans les onglets du navigateur. J'ai inclus plusieurs tailles pour assurer une bonne visibilité sur tous les appareils.

7 PARTIE VI : Servir l'application

7.1 Architecture de déploiement

Pour mettre en service **EcoTrace**, j'ai adopté une approche flexible qui facilite aussi bien le développement local que le déploiement en production. L'application repose sur Flask comme serveur web, avec une configuration adaptable selon l'environnement.

7.2 Test en local

7.2.1 Fichier de lancement (`run_app.py`)

J'ai mis en place un point d'entrée simple et efficace pour l'application :

```
from configs.settings import app

if __name__ == "__main__":
    app.run(debug=False)
```

Cette méthode présente plusieurs avantages :

— Simplicité : Un point d'entrée unique et clair

— Configuration centralisée : Toute la configuration est gérée dans le module `configs.settings`

— Sécurité : Le mode debug est désactivé par défaut pour une configuration plus sûre

7.2.2 Configuration pour le développement

Pour le développement local, il est possible d'utiliser :

```
if __name__ == "__main__":
    app.run(debug=True, host='127.0.0.1', port=5000)
```

Cependant, j'ai préféré conserver `debug=False` afin de rester au plus proche des conditions de production. Voici les étapes pour lancer l'application en local :

1. **Création de l'environnement virtuel et installation des dépendances** :

   ```
   python -m venv venv
   source venv/bin/activate

   pip install -r requirements.txt
   ```

2. **Lancement de l'application** :

   ```
   python run_app.py
   ```

3. **Accès à l'application** :

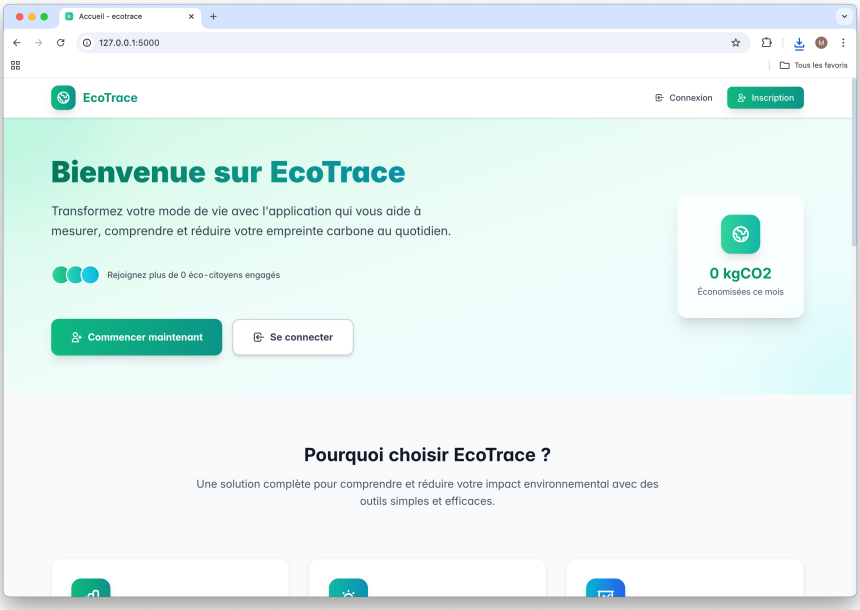
   ```
   http://127.0.0.1:5000
   ```

```
~/Documents/GitHub/ecotrace-l2 git:(main)  10 files changed, 30 insertio
python3 -m venv venv

~/Documents/GitHub/ecotrace-l2 git:(main)  10 files changed, 30 insertio
source venv/bin/activate

venv ~/Documents/GitHub/ecotrace-l2 git:(main)  10 files changed, 30 insertions(+
python3 run_app.py
 * Serving Flask app 'configs.settings'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production dep
loyment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

L'application est alors accessible à l'adresse `http://127.0.0.1:5000`



7.2.3 Mise en production : Déploiement sur PythonAnywhere (offre gratuite)

J'ai choisi PythonAnywhere pour déployer EcoTrace en production. Cette plateforme propose un hébergement Python gratuit, parfaitement adapté aux applications Flask. L'application est accessible à l'adresse : https://ecotrace.pythonanywhere.com

1. **Préparation du code**

```
# Vérification que tous les fichiers sont prêts
git add .
git commit -m "Préparation pour déploiement PythonAnywhere"
git push origin main
```

2. **Création du compte PythonAnywhere** :
   - Inscription sur pythonanywhere.com avec l'offre gratuite
   - Accès au tableau de bord PythonAnywhere
   - Ouverture d'une console Bash

3. **Upload du code**

```
# Dans la console PythonAnywhere
git clone https://github.com/codewithmpia/ecotrace-l2.git
cd ecotrace-l2
```

4. **Création de l'environnement virtuel et installation des dépendances**

```
# Dans la console PythonAnywhere
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

5. **Configuration WSGI : création du fichier `/var/www/ecotrace_pythonanywhere_com_wsgi.py`**

```
import sys
import os

# Ajout du chemin vers votre application
path = '/home/ecotrace/ecotrace-l2'
if path not in sys.path:
    sys.path.append(path)

# Import de l'application Flask
from configs.settings import app as application
```

6. **Configuration de l'application web**
   - Accéder à l'onglet "Web" dans le tableau de bord
   - Créer une nouvelle web app Flask
   - Définir le chemin source : `/home/ecotrace/ecotrace-l2`
   - Indiquer le fichier WSGI : `/var/www/ecotrace_pythonanywhere_com_wsgi.py`
7. **Configuration des fichiers statiques**

```
# Dans l'onglet "Static files"
URL: /static/
Directory: /home/ecotrace/ecotrace-l2/assets/static/
```

8. **Initialisation de la base de données**

```
# Dans la console PythonAnywhere
cd ecotrace-l2
python -c "from configs.settings import app, db;
app.app_context().push(); db.create_all()"
```

9. **Test et lancement**
   - Cliquer sur "Reload" dans l'onglet Web
   - Vérifier que l'application fonctionne à l'adresse fournie
   - Tester les fonctionnalités principales
10. **Avantages de PythonAnywhere pour EcoTrace**
    (a) Simplicité de déploiement :
    - Aucune configuration serveur complexe

— Interface web intuitive
— Support natif de Python et Flask

(b) Offre gratuite suffisante :
— 512 MB d'espace disque
— Bande passante adaptée à un projet étudiant
— Base de données SQLite incluse
— Domaine `pythonanywhere.com` fourni

(c) Maintenance simplifiée :
— Redémarrage facile via l'interface web
— Console intégrée pour les commandes
— Accès direct aux logs

EcoTrace fonctionne parfaitement sur PythonAnywhere et démontre qu'il est possible de déployer une application Flask complète et performante avec l'offre gratuite, tout en assurant une expérience utilisateur fluide et fiable.

8 Conclusion et perspectives

Pour conclure, ce projet EcoTrace ma permis de mettre en pratique de nombreuses compétences en développement web, de la conception de la base de données à la création dune interface moderne avec Flask et TailwindCSS.

Jai découvert limportance de structurer le code en modules pour faciliter la maintenance et lévolution de lapplication. Ce travail ma aussi sensibilisé aux enjeux de lécologie et à la façon dont le numérique peut aider chacun à agir concrètement.

Je suis fier davoir pu déployer une application fonctionnelle en ligne et je pense que ce projet pourrait être enrichi avec des fonctionnalités collaboratives ou des données en temps réel.

Cette expérience ma donné envie de continuer à développer des outils utiles pour la société et dapprofondir mes connaissances en programmation.

Le site est désormais accessible à l'adresse : https://ecotrace.pythonanywhere.com, et le code source est disponible sur GitHub : https://github.com/codewithmpia/ecotrace-l2.

9 Difficultés rencontrées

9.1 Difficultés techniques

9.1.1 Gestion des facteurs d'émission

- **Problème** : Intégration et structuration des données ADEME dans la base de données
- **Défis** :
  - Conversion des unités hétérogènes (km, kg, kWh, unités)
  - Catégorisation cohérente des activités
  - Gestion des valeurs manquantes ou incomplètes
- **Solution adoptée** : Création d'un système de facteurs d'émission pré-intégrés avec validation et fallbacks

9.1.2 Architecture des dropdowns personnalisés

- **Problème** : Les éléments `<select>` HTML natifs sont difficiles à styliser avec TailwindCSS
- **Défis** :
  - Maintien de l'accessibilité avec des composants personnalisés
  - Gestion des événements clavier (Tab, Échap, Entrée)
  - Synchronisation entre l'affichage et les valeurs de formulaire
- **Solution adoptée** : Développement de dropdowns entièrement personnalisés en JavaScript avec délégation d'événements

9.1.3 Calendrier personnalisé

- **Problème** : Les input de type `date` ne sont pas uniformément supportés et stylisés
- **Défis** :
  - Calcul correct des jours du mois et des premières/dernières semaines
  - Gestion des années bissextiles
  - Positionnement dynamique pour éviter les débordements de viewport
  - Localisation en français
- **Solution adoptée** : Implémentation d'un calendrier complet en JavaScript avec détection de positionnement automatique

9.2 Difficultés d'architecture

9.2.1 Séparation des responsabilités

- **Problème** : Éviter la duplication de code entre les vues et maintenir une logique métier séparée
- **Défis** :
  - Centralisation des calculs carbone sans créer de dépendances circulaires
  - Réutilisation des méthodes de validation
  - Gestion des erreurs cohérente à travers l'application
- **Solution adoptée** : Création du module `controllers` avec classes spécialisées (CarbonCalculator, RecommendationEngine)

9.2.2 Gestion des états et des erreurs

- **Problème** : Maintenir la cohérence des données et la robustesse face aux erreurs
- **Défis** :
  - Validation des données à plusieurs niveaux (frontend, backend, base de données)
  - Gestion des cas limites (données manquantes, utilisateurs sans activités)
  - Rollback automatique en cas d'erreur de transaction
- **Solution adoptée** : Implémentation de blocs try/catch systématiques avec fallbacks et messages utilisateur appropriés

9.3 Difficultés d'interface utilisateur

9.3.1 Responsive design complexe

- **Problème** : Adapter l'interface pour mobile, tablette et desktop avec des composants complexes
- **Défis** :
  - Transformation des tableaux en cartes pour mobile
  - Réorganisation de la navigation sur petits écrans
  - Maintien de l'utilisabilité des formulaires complexes
- **Solution adoptée** : Approche mobile-first avec composants adaptatifs et breakpoints TailwindCSS

9.3.2 Performance des graphiques

- **Problème** : Affichage fluide des visualisations Chart.js avec des datasets importants
- **Défis** :
  - Préparation des données côté serveur vs côté client
  - Gestion de la mémoire avec des graphiques multiples
  - Animations smooth sans impact sur les performances
- **Solution adoptée** : Préparation des données en JSON côté serveur et utilisation de Chart.js avec configuration optimisée

9.4 Difficultés de déploiement

9.4.1 Configuration PythonAnywhere

- **Problème** : Adaptation de l'application locale pour l'environnement PythonAnywhere
- **Défis** :
  - Configuration du fichier WSGI approprié
  - Gestion des chemins relatifs vs absolus
  - Installation des dépendances dans l'environnement contraint
  - Initialisation de la base de données en production
- **Solution adoptée** : Création d'un fichier WSGI dédié et adaptation des chemins statiques

9.4.2 Limitations de l'offre gratuite

- **Problème** : Optimisation pour les contraintes de l'hébergement gratuit
- **Défis** :
  - Limitation CPU et mémoire
  - Redémarrage automatique quotidien
  - Pas d'accès HTTPS personnalisé
- **Solution adoptée** : Optimisation du code, réduction des requêtes base de données, et acceptation des limitations pour un projet étudiant

9.5 Difficultés de données

9.5.1 Cohérence des facteurs d'émission

- **Problème** : Harmonisation des données ADEME avec la structure de l'application
- **Défis** :
  - Différences d'unités entre les sources
  - Granularité variable des données (générique vs spécifique)
  - Mise à jour et versioning des facteurs d'émission
- **Solution adoptée** : Création d'une base de données normalisée avec facteurs prévalidés et système de catégories cohérent

9.6 Leçons apprises

9.6.1 Planification et architecture

- Importance de la définition claire des modèles de données dès le début
- Nécessité d'anticiper les cas d'erreur et les états vides
- Valeur de la séparation stricte entre logique métier et présentation

9.6.2 Développement frontend

- Avantages des frameworks CSS utility-first comme TailwindCSS
- Complexité des composants interactifs accessibles
- Importance des tests sur différents navigateurs et appareils

9.6.3 Déploiement et production

- Différences significatives entre environnement de développement et production
- Importance de la documentation des étapes de déploiement
- Nécessité d'anticiper les contraintes d'hébergement

10 Sources et Références

10.1 Documentation technique

10.1.1 Framework backend

— Documentation Flask : https://flask.palletsprojects.com/en/2.3.x/
— Documentation SQLAlchemy : https://docs.sqlalchemy.org/en/20/
— Documentation Jinja2 : https://jinja.palletsprojects.com/en/3.1.x/

10.1.2 Extensions Flask

— Flask-Login : https://flask-login.readthedocs.io/en/latest/
— Flask-WTF : https://flask-wtf.readthedocs.io/en/1.2.x/
— Flask-SQLAlchemy : https://flask-sqlalchemy.palletsprojects.com/en/3.1.x/
— Flask-Minify : https://github.com/mrf345/flask_minify

10.1.3 Frontend et styling

— Documentation Tailwind CSS : https://tailwindcss.com/docs
— Documentation Chart.js : https://www.chartjs.org/docs/latest/
— Lucide Icons : https://lucide.dev

10.1.4 Hébergement et déploiement

— PythonAnywhere : https://www.pythonanywhere.com/
— Guide de déploiement Flask : https://help.pythonanywhere.com/pages/Flask/

10.2 Ressources de développement

10.2.1 Outils de développement

— Git et GitHub pour le versioning
— Visual Studio Code comme éditeur
— Chrome DevTools pour le débogage frontend
— Python virtual environments pour l'isolation des dépendances

10.2.2 Bonnes pratiques et patterns

— Architecture MVC avec Flask
— Patterns de sécurité web (CSRF, validation input)
— Responsive design principles
— Accessibility guidelines (WCAG)

10.3 Données et sources scientifiques

10.3.1 Facteurs d'émission carbone

— Base Carboneő de l'ADEME (Agence de l'Environnement et de la Maîtrise de l'Énergie)
— Données officielles françaises pour les calculs d'empreinte carbone
— Facteurs d'émission par secteur (transport, alimentation, énergie, consommation)

10.4 Inspiration et références design

10.4.1 Design system et UI/UX

— Material Design principles
— Apple Human Interface Guidelines
— Figma community resources
— Modern web design trends

10.4.2 Accessibilité

— WCAG 2.1 Guidelines
— WebAIM resources
— Inclusive design principles

10.5 Outils et technologies

10.5.1 Bibliothèques JavaScript

— Chart.js pour la visualisation de données
— Vanilla JavaScript pour les interactions
— Modern ES6+ features

10.5.2 CSS et styling

— Tailwind CSS utility-first framework
— CSS Grid et Flexbox pour les layouts
— CSS custom properties pour la théma

10.5.3 Développement et qualité

— Python PEP 8 style guide
— HTML5 semantic markup
— Progressive enhancement principles