Documentation technique du site d'avis d'établissements

Ce document technique détaille le fonctionnement du site web d'avis sur les établissements, développé avec Flask, SQLAlchemy, LeafletJS et déployé sur Render. Il explique le fonctionnement global, la structure technique, la sécurité, l'API, et donne des repères pour prise en main et maintenance.

0. Login et URL

L'URL de notre site est la suivante : https://projet-web-d6ri.onrender.com

Pour vous connecter en tant qu'utilisateur, vous pouvez utiliser l'adresse mail stagiaire@gmail.com avec le mot de passe stagiaire.

Pour vous connecter en tant qu'admin, vous pouvez utiliser l'adresse mail admin@gmail.com avec le mot de passe admin

1. Vue d'ensemble et Objectifs

Le projet permet à des utilisateurs de visualiser une carte interactive des établissements (restaurants, lieux, etc.), d'ajouter ou consulter des avis, et de gérer administrativement les données. Il repose sur un backend Flask avec gestion des sessions utilisateurs (authentification), une base SQL (MySQL ou équivalent), et une interface front riche en interactions JS (Leaflet pour la carte, AJAX pour les fiches, etc.).

2. Arborescence du projet

- /app : code principal de l'application (Python, templates, static)
  - __init__.py : initialisation Flask + extensions
  - models.py : modèles SQLAlchemy (User, Etablissement, Category, Retour)
  - routes.py : toutes les routes et vues Flask
  - templates/ : HTML Jinja (pages principales, fragments AJAX, admin, etc.)
  - static/css : CSS (home.css, admin.css…)
  - static/js : JS (home.js…)

- config.py : configuration Flask (clé secrète, SQL…)
- requirements.txt : dépendances Python
- /venv : environnement virtuel (non versionné)

3. Base de données & Modèles

La base de données comporte 4 tables principales :
- utilisateurs : gère l'authentification, les droits (admin ou simple user)
- etablissements : infos sur les établissements (nom, adresse, coords, catégorie)
- categories : liste de toutes les catégories possibles
- retours : stocke chaque avis (note, commentaire, date, auteur, établissement)
Les modèles sont définis dans app/models.py. SQLAlchemy est utilisé partout (ORM).

4. Authentification & sécurité

L'accès à la plupart des routes est contrôlé par Flask-Login :
- Toute action d'ajout/suppression/modification d'avis nécessite d'être connecté.
- L'administration (dashboard, gestion users/étabs/catégories) est réservée aux admins, avec un décorateur @admin_required.
- Les routes publiques (home, visualisation carte, consultation avis) sont accessibles même sans connexion (voir section 6).
La connexion/déconnexion fonctionne via des sessions sécurisées. Les mots de passe sont stockés hashés.

5. Fonctionnement du Backend (routes)

- / : redirige vers /home
- /home : carte interactive, affichage d'établissements (requête SQL puis passage en JSON pour JS)
- /login /logout /add_user : gestion des utilisateurs
- /dashboard, /etablissements, /categories… : administration (admins uniquement)
- /add_etablissement, /edit_etablissement, etc. : ajout/modif/suppression d'étabs
- /fiche_etablissement_fragment/<id> : fragment AJAX détaillé (infos, avis, formulaire)
Toutes les routes nécessitant un rôle sont décorées par @login_required et/ou @admin_required.

6. Fonctionnement du frontend et JS

- La carte interactive (Leaflet) reçoit via JS un tableau d'établissements, puis crée les marqueurs selon leur catégorie.
- Les contrôles (zoom, recentrage…) sont customisés.
- Les recherches/filtrages se font en JS (pas d'appel AJAX ici).
- L'autocomplétion sur la recherche s'active en JS.
- Le clic sur un marqueur ouvre un panneau latéral chargé dynamiquement (AJAX) avec la fiche détaillée (avis, formulaire d'avis si connecté).
- Les avis ne peuvent être ajoutés ou modifiés que si l'utilisateur est connecté (sinon le bouton ou le formulaire n'apparaît pas).
- Sur mobile, le responsive est géré en CSS (media queries, vh, em, vw).

7. Fonctionnalités admin

Un administrateur peut :
- Ajouter, modifier ou supprimer utilisateurs, établissements, catégories (avec contrôles d'accès via @admin_required)
- Consulter toutes les données via des dashboards filtrés
- Ajout d'établissement : la latitude/longitude peuvent être auto-remplies par appel JS à l'API Nominatim (OpenStreetMap) depuis l'adresse tapée.

8. Déploiement sur Render

- Les fichiers sont poussés sur GitHub (dépôt privé ou public).
- Le projet est connecté à Render, qui build le conteneur Flask via requirements.txt.
- Les variables d'environnement (clé SQL, SECRET_KEY…) sont configurées côté Render.
- La base de données est reliée par une URL distante (Render PostgreSQL ou MySQL).
- Un script de démarrage lance app.py (attention au port : 10000 sur Render, 5001 en local).

9. Notes techniques et prise en main

- Pour installer toutes les dépendances : pip install -r requirements.txt (dans un venv).
- Lancer le projet localement : flask run ou python app.py
- Structure MVC : la logique est séparée entre modèles, vues (HTML/Jinja), routes (contrôleurs).
- Le code est commenté pour faciliter la maintenance.
- Pour rendre le site totalement public, il suffit d'enlever @login_required sur /home et les routes de consultation. Mais les routes de création/modif d'avis, d'admin, restent protégées.

10. Pour aller plus loin / évolution

- Ajouter la pagination sur les listes d’avis ou d’établissements
- Ajouter une API REST publique (read only) pour consultation hors site
- Personnaliser le design (Dark mode, nouveaux styles)
- Améliorer la gestion des droits (rôles, permissions avancées)