Université Paris Nanterre

Licence MIASHS première année

Rapport de projet informatique

Aurore-Boréale : assistant IA pour les étudiants de l'Université Paris Nanterre

Projet réalisé du 2025 au 2026

Membres du groupe

Nom Prénom
Nom Prénom

Remerciements

Nous remercions l'équipe enseignante de l'UE de programmation web pour l'encadrement du projet, ainsi que les personnes qui ont testé notre application et nous ont fourni des retours utiles.

Table des matières

1 Introduction

Ce rapport présente le projet Aurore-Boréale, un assistant conversationnel destiné aux étudiants de l'Université Paris Nanterre. Le site internet permet de poser des questions sur des sujets liés à l'université (inscription, scolarité, vie de campus, informations pratiques, ...) et d'obtenir des réponses générées par une intelligence artificielle.

L'objectif principal est de fournir un point d'entrée unique et simple d'utilisation pour aider les étudiants à trouver rapidement les informations dont ils ont besoin, tout en conservant un historique personnalisé des questions et réponses pour chaque compte utilisateur.

Lien vers le code source

Le code source complet du projet Aurore-Boréale est disponible sur le dépôt GitHub public suivant :
`LIENS`

2 Environnement de travail

Le projet a été développé principalement sous Ubuntu/WSL avec l'éditeur Visual Studio Code. Le code est hébergé sur un dépôt GitHub public, ce qui a facilité le travail collaboratif et la sauvegarde de l'historique des modifications.

Nous utilisons un environnement virtuel Python (`venv`) pour isoler les dépendances (Flask, client de l'API d'IA, etc.) ainsi qu'une base de données SQLite locale pour stocker les utilisateurs et les échanges avec l'assistant.

3 Description du projet et objectifs

3.1 Objectifs fonctionnels

Le projet Aurore-Boréale a pour objectif de fournir aux étudiants de l'Université Paris Nanterre un assistant numérique capable de répondre à leurs questions sur la scolarité, l'inscription, la vie de campus et les services proposés par l'université. Le site internet doit être simple d'utilisation et accessible depuis un navigateur web. Dans notre cas un site local.

Les principaux objectifs fonctionnels sont :

— Permettre à un étudiant de créer un compte, se connecter et se déconnecter ;
— Offrir une interface de chat pour poser des questions à l'assistant IA ;
— Enregistrer les questions et les réponses dans une base de données, en les associant à l'utilisateur connecté ;
— Afficher un historique des échanges, avec la date, la question, la réponse et le pseudo de l'utilisateur ;
— Permettre à chaque utilisateur d'effacer l'intégralité de son historique de questions/réponses ;
— Avoir une IA qui répond bien aux questions poser par l'utilisateur.

3.2 Scénario d'utilisation

Un étudiant arrive sur la page d'accueil du site puis se connecte, s'inscrit ou peux rester sans utilisateur. Il est sur la page de chat et pose une question liée à l'université (par exemple les horaires de la bibliothèque ou la procédure pour obtenir une attestation de scolarité). L'assistant IA génère une réponse, qui est affichée dans l'interface et enregistrée dans la base de données. L'étudiant peut consulter l'historique de ses questions, vérifier les réponses précédentes et, s'il le souhaite, effacer tout son historique via un bouton dédié. Mais pas son historique dans la base de données.

4 Bibliothèques, outils et technologies

Les principales technologies utilisées sont :
— **Python** pour la logique serveur ;
— **Flask** comme framework web pour gérer les routes, les sessions et les templates ;
— **SQLite** pour la base de données (tables utilisateurs, questions, réponses) ;
— **HTML / CSS** pour les pages et la mise en forme (barre de navigation, historique, formulaires) ;
— **API d'IA** (modèle de langage Mistral) pour générer les réponses aux questions des étudiants ;
— **Git et GitHub** pour le suivi de version et l'hébergement du code source.

5 Bibliothèques, outils et technologies

Le projet Aurore-Boréale s'appuie sur les technologies suivantes :
— **Python** : langage principal pour la logique serveur.
— **Flask** : framework web léger permettant de définir les routes (`/`, `/login`, `/signup`, `/history`, `/api/chat`, etc.), de gérer les sessions et de rendre les templates HTML.
— **SQLite** : base de données relationnelle embarquée utilisée pour stocker les utilisateurs, les questions et les réponses de l'assistant.
— **HTML / CSS** : construction de l'interface utilisateur (pages d'accueil, formulaires de connexion et d'inscription, page de chat, historique).
— **JavaScript** : envoi asynchrone des messages au serveur (`/api/chat`) et mise à jour dynamique de l'interface de discussion.
— **API d'IA** : modèle de langage accessible via une API (type Mistral), utilisé pour générer les réponses aux questions des étudiants à partir d'un prompt construit par le backend.
— **GitHub** : gestion de version du code source et dépôt public du projet, contenant également un répertoire `rapport/` avec les fichiers LaTeX et les images nécessaires à la compilation du présent document.
— **Google Docs** : Pour nos réunions hebdomaires, et le suivi de différents dossiers.

6 Fonctionnement de l'IA utilisée

L'application Aurore-Boréale utilise une API d'intelligence artificielle de type modèle de langage (LLM), similaire à un modèle Mistral accessible via requêtes HTTP depuis le serveur Flask. Pour chaque question posée par un étudiant, le backend construit

un *prompt* textuel qui décrit le rôle de l'IA, le contexte universitaire et l'historique récent des échanges de cet utilisateur.

Le prompt contient notamment :

— des instructions expliquant que l'IA doit répondre uniquement à des questions en lien avec l'Université Paris Nanterre (inscription, scolarité, vie de campus, services aux étudiants) ;
— un rappel du ton attendu : réponses en français, claires, structurées, sans emoji ;
— plusieurs paires question/réponse récentes issues de la base de données pour l'utilisateur connecté ;
— la nouvelle question que l'étudiant vient de saisir.

Le serveur envoie ce prompt à l'API d'IA et reçoit en retour un texte de réponse. Cette réponse est nettoyée (suppression d'éventuels emojis et espaces superflus) puis enregistrée dans la table `reponses`, liée à la question correspondante dans la table `questions`. Lors d'une prochaine question du même compte, l'application récupère les échanges précédents de cet utilisateur pour les réinjecter dans le prompt, ce qui donne à l'assistant une mémoire personnalisée par étudiant.

7 Comment l'IA nous a aidé dans le projet

Durant la réalisation du projet Aurore-Boréale, nous avons utilisé l'IA comme outil d'assistance tout au long du développement.

7.1 Aide à la conception

L'IA nous a aidés à clarifier le cahier des charges et à structurer l'application. Elle a proposé une organisation des routes Flask (`/login`, `/signup`, `/history`, `/api/chat`) ainsi qu'un modèle de base de données séparant les tables `users`, `questions` et `reponses`. Ces suggestions ont servi de base que nous avons ensuite adaptée aux contraintes du projet.

7.2 Aide au développement et au débogage

Nous avons sollicité l'IA pour générer ou compléter certains fragments de code (requêtes SQL, gestion des sessions, gabarits HTML/Jinja) et pour analyser des messages d'erreur. Elle a par exemple permis d'identifier une requête d'insertion non exécutée ou un problème de chemin de base de données. Chaque extrait proposé a été relu, testé et corrigé si nécessaire avant d'être intégré au projet.

8 Travail réalisé

Cette section présente les fonctionnalités prévues au départ, celles qui ont été effectivement implémentées et la répartition réelle du travail au sein du groupe.

8.1 Fonctionnalités prévues

Au début du projet, nous avions identifié les fonctionnalités suivantes :

— système d'inscription, de connexion et de déconnexion des utilisateurs ;
— interface de chat avec l'assistant IA ;

— enregistrement de toutes les questions et réponses dans une base de données ;
— historique personnel par utilisateur avec mémoire de l'IA ;
— possibilité pour un utilisateur d'effacer l'intégralité de son historique ;
— intégration d'informations officielles de l'université (horaires, contacts, etc.).

8.2 Fonctionnalités réalisées

Dans la version actuelle de l'application, nous avons réalisé :
— la création de compte, la connexion et la déconnexion avec gestion de session ;
— la page de chat permettant d'envoyer une question à l'IA et d'afficher la réponse ;
— l'enregistrement des questions et réponses en base, associées à l'utilisateur connecté ;
— une page d'historique affichant les questions, les réponses, la date et le pseudo ;
— une mémoire de l'IA limitée aux échanges de l'utilisateur courant ;
— un bouton permettant à chaque utilisateur de supprimer tout son historique.

8.3 Fonctionnalités non réalisées et raisons

Certaines fonctionnalités n'ont pas pu être terminées, par exemple :
— l'intégration complète de données mises à jour automatiquement depuis le site de l'université ;
— une interface d'administration avancée pour analyser les questions fréquente ;
— Difficultés lié au scrappring.

Ces éléments ont été jugés moins prioritaires compte tenu du temps disponible et de la nécessité de stabiliser d'abord le coeur du système (authentification, base de données, intégration de l'IA).

9 Difficultés rencontrées

Nous avons rencontré plusieurs difficultés techniques au cours du projet. La première concernait la gestion correcte de la base de données SQLite : certaines requêtes d'insertion n'étaient pas exécutées (erreurs de syntaxe ou d'appel de fonction), ce qui empêchait l'historique de se mettre à jour.

Nous avons également eu des problèmes liés aux sessions Flask (gestion de `session["user`$_i d$"]$et à la

10 Bilan

10.1 Conclusion

Le projet Aurore-Boréale nous a permis de mettre en pratique les notions de développement web vues en cours : création d'une application Flask, utilisation d'une base de données, gestion d'utilisateurs et intégration d'un service externe d'intelligence artificielle. Nous avons également découvert les enjeux spécifiques liés à l'utilisation d'un modèle de langage dans une application destinée à de vrais utilisateurs (qualité des réponses, cadre d'usage, stockage des échanges).

10.2 Perspectives

Plusieurs améliorations sont envisageables pour la suite :

-- enrichir la base de connaissances en important automatiquement des informations à jour depuis le site de l'université ;
-- améliorer l'interface graphique et l'expérience utilisateur (design, accessibilité, version mobile) ;
-- ajouter une interface d'administration pour analyser les questions fréquentes et ajuster les consignes données à l'IA ;
-- expérimenter d'autres modèles d'IA ou combiner l'IA avec une base de FAQ structurée afin de renforcer la fiabilité des réponses.

11 Bibliographie

[LAM94] L. Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley, 1994.

12 Webographie

[DOCFLASK] https://flask.palletsprojects.com/

[SQLITE] https://www.sqlite.org/docs.html

[MISTRAL] https://docs.mistral.ai/

13 Annexes

Annexe A: Cahier des charges

Cette annexe rappelle les exigences initiales du projet (fonctionnalités demandées, contraintes techniques, etc.).

Annexe B: Exemple d'exécution du projet

Cette annexe contient un exemple complet d'utilisation de l'application avec des captures d'écran (page d'accueil, connexion, chat avec question et réponse, historique, etc.).

Annexe C: Manuel utilisateur

Cette annexe explique comment lancer le projet (cloner le dépôt GitHub, créer l'environnement virtuel, installer les dépendances, lancer le serveur Flask) et comment utiliser les principales fonctionnalités.