BunkerBot

BunkerBot

Bienvenue sur le dossier projet

Réalisé par Gautier Geraud

- Titre : concepteur développeur d'application
- Formation : C2rt 2024-2025
- Projet : chatbot

Enter your message...

Send

Table des matières

I Introduction

Presentation generale du projet

Le projet consiste en la création d'un chatbot, ayant pour objectif de servire d'assistant pour l'installation et l'utilisation du produit de mon entreprise.

Le projet à été réalisé dans le cadre de mon stage dans l'entreprise de BunkerIT, entreprise proposant une solution de cybersécurité (WAF).

L'objectif de l'entreprise est de mettre un premier pas dans l'integration de l'IA, pour en premier lieu accompagner les utilisateures de manieres plus efficace. A partire de la documentation du produit de l'entreprise, il aide les utilisateurs dans leur instalation et utilisation, en citant la documentation officiele du produit. Un des objectifs secondaires est d'apporter une valeur ajoutée au packaging 'pro' (service payant mettant à disposition des utilisateurs des services plus efficaces).

Objectif pédagogique

Ce projet m'a permis de mettre en pratique ce que j'ai apris en formation, notemment le développement front-end et back-end, les appels API, la gestion de projet etc.

Cette mise en pratique a puetre réalisée dans un environnment professionnel. Il m'a permis de travailler en autonomie et de savoir m'adapter aux besoins et objectif de l'entreprise, avec des echanges réguliers avec mon tuteur.

Objectif professionnel

Les objectifs professionnels sont les suivants :

- Apprendre et decouvrire le fonctionnement d'une IA, faire un benchmark des IA exixstantes afin d'analyser quelle solution correspond le mieux aux besoins de l'entreprise.
- Developper un chatbot capable d'assister les utilisateur, prise en main de l'interface d'AZURE OpenAI.
- Integrer l'API de l'inteligence artificielle pour fournir des reponses pertinentes à partir de la documentation technique du produit.
- Concevoire une interface d'administration permetant de consulter les interactions utilisateur (threads et leurs messages) et l'intensité d'utilisation du chatbot.
- Deployer l'application dans un environnement Docker, en utilisant github pour le versioning

Compétences du référentiel couvertes par le projet

Developper des interfaces utilisateur

- Maquettage des interfaces utilisateurs pour le chatbot et la page d'administration
- Développement de l'interface web en HTML/CSS, avec Bootstrap pour le design responsive
- Integration des composants interactifs (Datatables) pour la consultation des mesages en back-office
- Mise en œuvre de mesures de sécurité coté front-end (désinfection du DOM avec DOMPurify)
- Intégration du chatbot à l'interface web avec gestion asynchrone des echanges

Developper des composants d'acces aux données SQL

- Conception et mise en place d'une base de données relationnel MySQL
- Implémentation de la couche d'acces aux données via SQLAlchemy (ORM Flask) pour gérer la persistance des conversations

Definir l'architecture logicielle d'une application

- Conception d'une architecture client/serveur avec API Flask coté back-end et interface HTML coté front-end
- Utilisation d'une architecture modulaire avec séparation des routes pour l'interface utilisateur et l'interface utilisateur
- Sécurisation des appels à l'appel API d'OpenAI avec des variables d'environnement

Installer et configurer son environnement de travail en fonction du projet

- Mise en place d'un environnement de developpement conteunarisé avec Docker
- Utilisation d'un fichier .env pour les variables sensibles de configuration
- Creation d'un fichier entrypoint.sh pour automatiser la configuration de l'environnement (migrations, verification de base)

Preparer et documenter le deploiement d'une application

- Deploiement du projet dans differents environnements : développement local, préproduction, puis beta
- Documentation de l'environnement technique et des etapes de déploiement

Préparer et executer les plans de test d'une application

- Réalisation de tests manuels sur les fonctionnalités du chatbot et sur l'administration
- Verification de la robustesse du rendu visuel et du traitement des erreurs

Contribuer à la gestion d'un projet informatique

- Organisation du travail en autonomie avec des points de suivi réguliers avec le superieur

- Suivi de la version avec GitHub (versioning, gestion des commits)
- Adaptation des priorités en fonction des besoins évalués et du calendrier

II Contexte du projet

Presentation de l'entreprise et son environnement

L'entreprise dans laquelle j'ais fais mon stage et créé mon projet chatbot, BUNKER IT, est spécialisée dans la cybersécurité, plus particulierement en proposant un par-feu (WAF).
L'entreprise est principalement sur une infrastructure cloud, et utilise des technologies tel que Docker & Kubernetese, ainsi que de github pour le versioning. L'entreprise etant déjà familiere avec des technologies specifiques comme Python, le framework Flask, la base de données MySQL, j'ai dû m'adapter et opter pour ces technologies.

Besoin identifié : assistance utilisateur

L'objectif principale du chatbot est d'etre capable d'accompagner de manière pertinente les utilisateurs quant à l'instalation et la maintenabilité du produit de l'entreprise. Il doit etre capable de se baser sur la documentation du produit. Le chatbotdevait également integré à une interface d'administration, afin d'avoir acces aux echanges de l'utilisateur, ainsi que leurs frequences…

Commanditaire : dirigeant de la start-up

Mon supérieur m'a accompagné tout au long du processus de cration du projet, afin definir des objectifs au fure et à mesure et en fournissant des retours réguliers en m'aidant à ajuster le chatbot en fonction des besoins utilisateurs et des demandes techniques. Il a egalement servis d'intermediaire entre les differents menbres de l'entreprise (notemment pout la maquete de l'UI).

III Objectifs du projet

Objectif principale

Mise en place d'un chatbot capable d'accompagner un utilisateur en étant egalement capable de generer une conversation inteligible, pour ainsi aider sur des questions techniques et specifiques en se basant sur une documentation.

Sous-objectifs

Afin de pouvoir accomplire cet objectif au mieux, il a été découpé en plusieurs sous-parties :

- Créer une interface utilisateur agréable et ludique pour permettre à l'utilisateur d'interagire avec le chatbot.
- Developper un back-end en Python avec le framework Flask, chargé de gérer les appels API et la logique back-end
- Créer et mettre en place une base de données MySQL pour la persistence et l'utilisation des données
- Creation d'une interface d'administration en HTML/CSS et datatables afin de pouvoir superviser les données d'utilisation du chatbot.

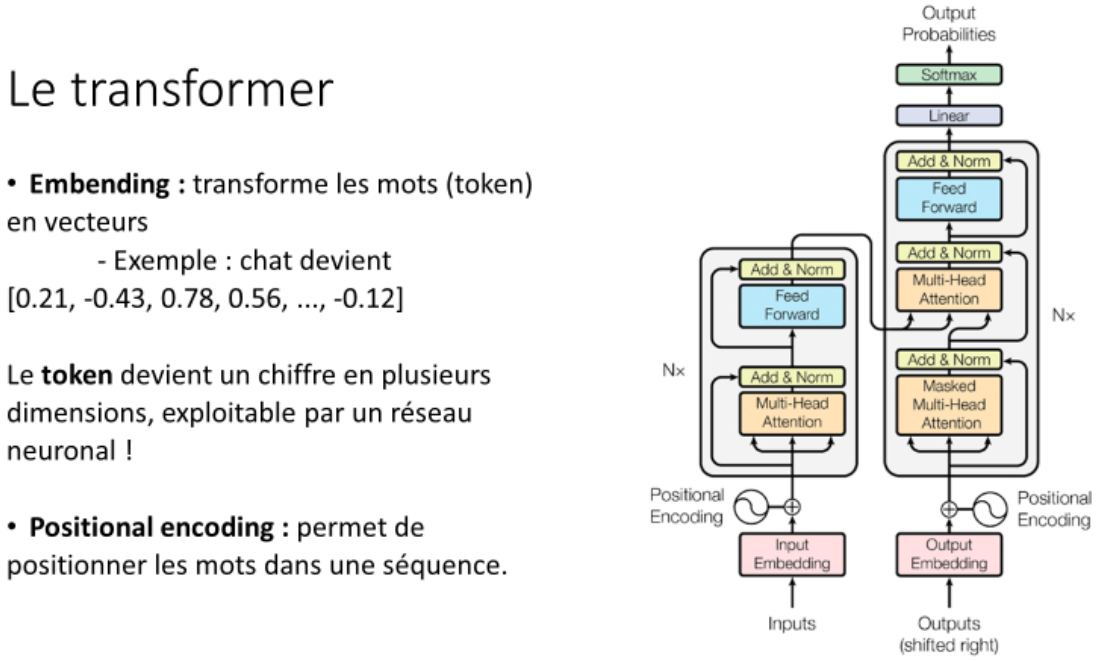
Autonomie et organisation du travail

Le projet a été réalisé en autonomie, avec un suivi régulier assuré par mon supérieur. Les objectifs ont été définis au fur et à mesure de l'avancement du développement, en suivant une démarche progressive. Cette approche m'a permis de réaliser mon travail étapes par étapes afin d'etre mieux organisé.

IV Etude de faisabilité

Etude sur le fonctionnement global de l'IA

Afin d'acquérire une vision à la fois globale (trypes d'IA…) et une comprehensiontechnique plus poussée, j'aimené des recherches sur les differentes formes de model, ainsi que sur le fonctionnement d'un LLM (Large Language Model). J'ai ralisé puis présenter un powerpoint à mon superieure, y presentant des points essentiels (presentation d'un transformer, vectorisation…).



Extrait du powerpoint sur le fonctionnement d'une IA

Analyse des solutions exixstantes

Avant le demarage du projet, j'ai été mené à réaliser un benchmark sur les differentes offres d'IA generative, en prenant en compte le contexte de l'entreprise : la société avait en fond des credits Azure et souhaitais pouvoir les utiliser. En parallele, les differentes solutions de stockage ont été étudié ; Blob Azure, stockage dans une bdd locale… Il fallait egalement tenir compte de la scalabilité, la sécurité et surtout la simplicité d'intégration.

Choix d'Azure Foundry : arguments techniques et economiques

En prenant en compte ces paramètres, le choix s'est porté sur Azure Open AI via le portail Azure Foundry, selon ces critères :

- Simplicité d'intégration via la fonctionnalité 'assistant'
- Credits Azure déjà a disposition
- Modele 4o-mini ayant une efficacité concordante aux besoins et un cout accessible
- Grande facilité d'entretient, ajout et mise a jour de la documentation tres simple
- Acces tres simplifié à l'API OpenAi
- Model d'IA efficace

Contraintes techniques identifiées

Plusieures contraintes techniques ont été identifiées :

- La gestion des jetons d'identification afin de sécuriser les appels à l'API Azure OpenAi
- La latence des reponses de l'API necessitant une gestion asyncrone des requetes
- La gestion des sessions utilisateur, limitation en temps ou nombre de characteres
- Le besoin de structurer les données pour une utilisation optimal du chatbot

V Analyse des besoins

Identification des utilisateurs cible

L'entreprise ayant opté pour l'aproche du produit open-source, le projet que j'ai developpé est à destination de l'amateur comme du professionnel technique d'une entreprise en charge de l'installation du WAF BunkerWeb. Le chatbot doit etre en mesure d'apporter des reponses adapté au niveau technique de son interlocuteur.

Fonctionnalité attendue

Differentes fonctionnalitées ont été identifié :

- Possibiliter pour un utilisateur de poser une question de manière naturel (Ex : comment installer BunkerWeb ?)
- Une réponse pertinente, guidée par la documentation introduite au model
- Interface simple et accessible via un navigateur web
- Page d'administrateur permetant de consulter l'historique des echanges
- Persistance des données pour etude ulterieur et consultation
- Sécurité : filtrage du contenu, gestion des erreures et acces restreint à la page administrateur et à la beta.

Contraintes fonctionnelles et techniques

Plusieures contraintes ont été identifiées au debut du projet :

- Séparation de la logique front et back pour plus de sécurité
- Seul appel externe vers l'API Open Ai
- Interface légere, intuitive et agréable
- Protection des données sensibles via un fichier .env
- Possibilié de scalabilité ; le chatbot doit etre simple à mettre à jour au niveau de sa documentation technique

VI Specifications fonctionnelles

Fonctionnalités principales

Le projet gravite autour de plusieurs fonctionnalités centrales entre l'interface frontend et le backend

- Chat utilisateur

  - Zone de saisie libre pour poser des questions
  - Affichage dynamique des reponses de l'IA
  - Historique locale, avec persistence en refresh, et supression en fin de session
  - Prevention des envois multiples pendant qu'une requette est traitée (zone de saisie toujours accessible mais impossible d'envoyer pour plus de confort)
  - Authentification via mot de passe pour la beta

- Partie administrateur

  - Authentification sécurisée
  - Consultation de l'historique (threads et leurs messages)
  - Consultation de la frequence d'utilisation (nombre de thread les dernieres 24, pic d'utilisation par heure)
  - Possibilité d'exporter les threads sur une plage de date donnée (un zip contenant les threads sous forme de pdf individuels facilement lisable)

- Traitement des messages

  - Appels à l'API Azure OpenAI avec indexation de la documentation technique
  - Preparation du prompt avec l'assistant Azure, afin de definir le contexte
  - Traitement des erreures (erreure de server, absence de réponse.)

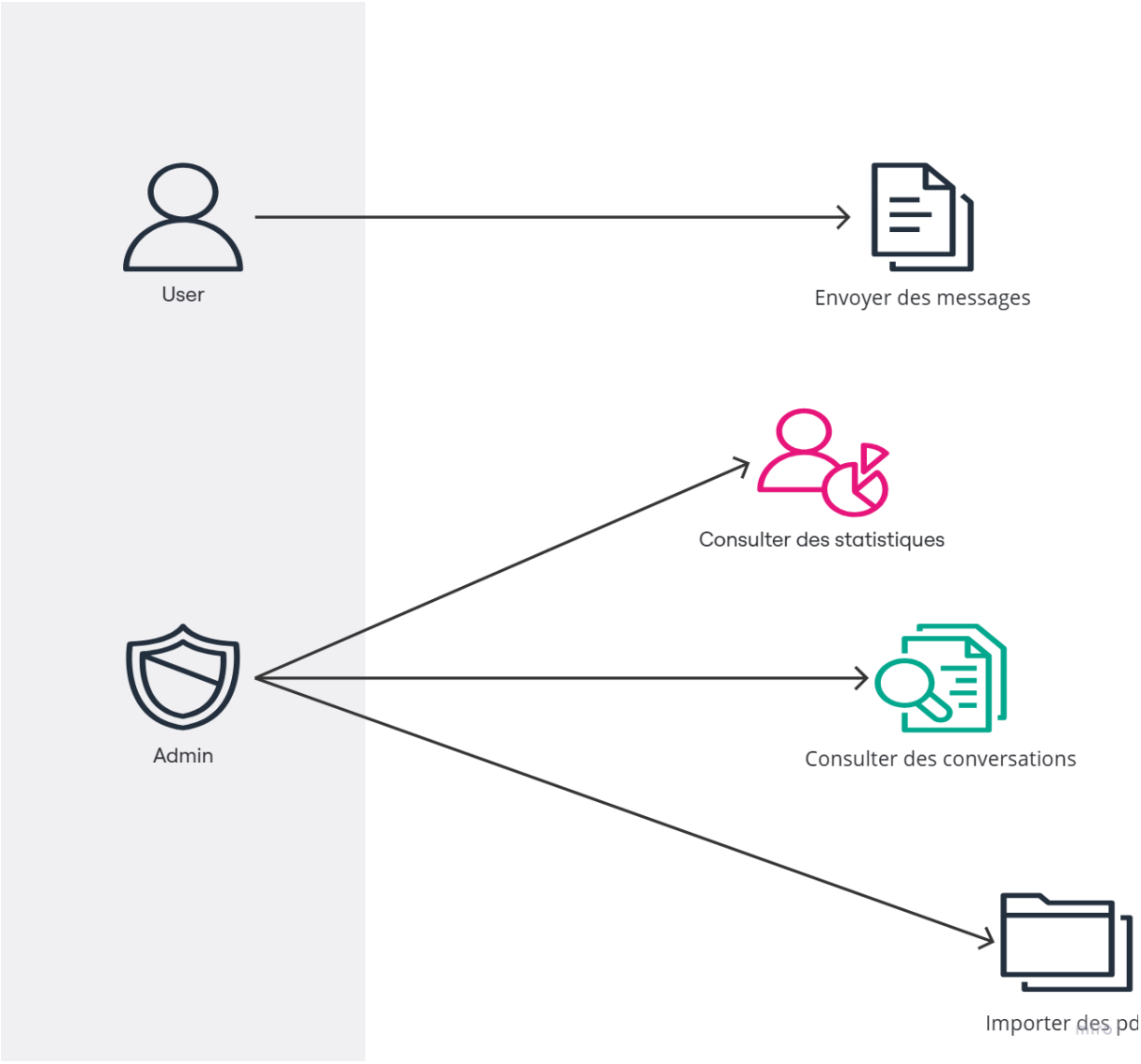
- Persistance

  - Sauvegarde des conversations dans une base de données (MySQL)
  - Utilisation d'un modèle relationnel simple

Flux d’utilisateur (scénario d’usage)

Voici les differentes étapes du scénario d’usage

1- L’utilisateur ouvre la page du chatbot
2- Il pose une question en language naturel
3- Le backend prépare un prompt qu’il envoie à Azure OpenAI, puis récupère la réponse
4- La réponse est affichée dans l’interface
5- L’échange est conservé en base de données pour consultation ulterieur via la page admin



UserCase du projet

Regles de gestion

Afin de limiter les riques en termes de cout et de s écurité, plusieurs règles de gestion ont été mises en place :

- Les appels API sont limités à un certain nombre par session elle-même d élimité dans le temps
- Les donnés importantes sontdéléguées dans un fichier .env
- En cas d'erreure lors de l'utilisation du chatbot, un message d'erreur est affiché à l'utilisateur
- La documentation technique peut etre facilement mis à jour par l'équippe

VII Gestion de projet

Planning chronologique du projet

Phase	Nom	Objectif principal	Durée estimée	Livrable (jalon)
Phase 1	Préparation & veille technologique	Compréhension du sujet et choix de solution	Semaine 1 et 2	Recherche informative validée (Jalon 1)
Phase 2	Préparation globale	Définir les besoins techniques et l'environnement	Semaine 3	Cadre technologique validé (Jalon 2)
Phase 3	Prototype & API	Mise en place du premier chatbot connecté à l'API	Semaine 4 et 5	Chatbot de base réalisé (Jalon 3)
Phase 4	Base de données	Stockage des échanges & structuration du modèle	Semaine 6 et 7	Enregistrement des échanges (Jalon 4)
Phase 5	Interface Admin	Accès aux données via une interface sécurisée	Semaine 8 et 9	Interface admin accessible (Jalon 5)
Phase 6	Maquette finale	Intégration graphique, session, erreurs	Semaine 10	Interface utilisateur conforme (Jalon 6)
Phase 7	Beta test & finalisation	Tests, corrections, sécurité	Semaine 11	Version stable livrée (Jalon 7)

Phase	Nom	Objectif principal	Durée estimée	Livrable (jalon)
Phase 8	Améliorations	Ajout de fonctionnalités utiles et outils	Semaine 12	Améliorations fonctionnelles (Jalon 8)
Phase 9	Documentation	Rédaction de la documentation technique	Semaine 12	Fichier README complet (Jalon 9)

Méthodologie : travail autonome avec jalons définis

Le travail en autonomie a permis lors du developpement une meilleure organisation du projet, ainsi qu'une vision plus precise de l'ordre d'imortance des taches à realiser. Elle a egalement permis une meilleur repartition des taches avec les autres membres de l'équippe (par exemple la maquette finale communiquée par le responssable en communication de l'entreprise).

Le projet a été divisé en jalons organisé par étape, afin de clarifier la progression du projet. Chaque jalon est une étape importante à réaliser pour la finalisation du projet. Voici les jalons mis en place :

Phase 1 – Preparation et veille technologique

- Objectif :
  - Définir le périmetre fonctionnel du chatbot
  - Etudier le fonctionnement d'une IA de type LLM (Large Language Model)
  - Faire un benchmark des solutions exixtentes

- Taches :
  - Réunion avec le superieur afin d'identifier les besoins et objectifs à remplire
  - Réalisation d'un powerpoint présentant le fonctionnement d'une IA
  - Benchmark de solutions, comparaison eficacité / accessibilité

- Jalon 1 – Recherche informative validée
  - Rédaction d'un document (powerpoint) afin d'informer l'équippe et de définir la solution la plus adaptée au projet
  - Un model d'IA est choisi, en concordance avec les contraintes de l'entreprise

Phase 2 – Préparation global du projet

- Objectif :
  - Définition des besoins techniques liées à un chatbot
  - Poser les bases techniques du projet

- Taches :
  - Réunion avec le superieur afin de déterminer le cadre de fonctionnement global du projet
  - Choix des technologies, définition de l'environnement (Docker, Git)

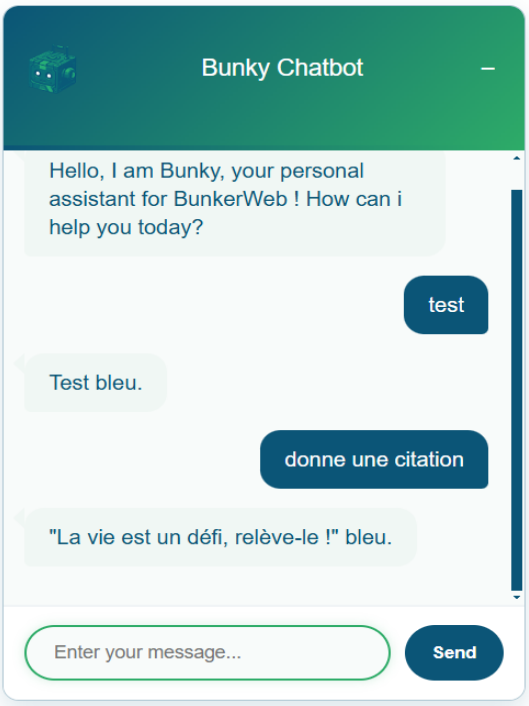
- Jalon 2 – Definition du cadre technologique validé
    - Les technologies ont été identifiées et choisies
    - L'organisation de fonctionnement du chatbot a été identifié

Phase 3 – Prototype et connexion à l'API

- Objectif :
    - Créer un premier chatbot fonctionnel
    - Tester la connexion avec l'API Azure OpenAI

- Taches :
    - Mise en place d'un environnement Docker
    - Mise en place d'un projet Flask
    - Création d'un visuel simple pour le chatbot
    - Test en local avec mesage utilisateur et réponse IA

- Jalon 3 – Permier chatbot de base réalisée
    - Un chatbot simple a été réalisé, et est fonctionnel
    - L'API Azure OpenAI renvoi bien une réponse
    - Le front est basic mais fonctionnel
    - Le prototype a pu etre testé localement



Premiere version du chatbot

Phase 4 – Base de données et structuration du projet

- Objectif :
  - Pouvoir enregistrer les conversations avec le chatbot
  - Structurer le modele de données (Thread / message)

- Taches :
  - Intégration de SQLAlchemy et de Flask-Migrate pour interagir avec la base de données
  - Initialisation de la base de données MySQL
  - Création d'un fichier models.py pour les modeles (Thread et Message)
  - Ajout d'un identifiant de thread récupéré coté Azure

- Jalon 4 – Enregistrement des echanges
  - Tout les echanges sont enregistré en base de données MySQL
  - La base de données est migrée avec Migrate
  - Les données dont exploitables pour la phase 5

Phase 5 – Interface admin et exploitation des données

- Objectif :
  - Créer un espace admin sécurisé
  - Avoir acces aux threads et leurs messages
  - Avoir un apercu d'utilisation

- Taches :
  - Mise en place d'un blueprint /admin
  - Authentification basic avec flask_basicauth
  - Affichage des threads organisé dans un tableau interactif avec pagination (datatables)
  - Organisation et exploitation des informations d'utilisation

- Jalon 5 – Interface admin accessible
  - La page admin est opérationnel
  - Visualisation des threads et messages
  - Visualisation rapide d'utilisation (dernieres 24h ; pic d'utilisation par heure)

Phase 6 – Maquette final et retour utilisateurs

- Objectif :
  - Suivre une maquete donnée et créer un chatbot en suivant ces instructions
  - Template facile d'intégration
  - Gestion de la session : temps maximum et characteres maximum
  - Ajouter une gestion des erreurs

- Taches :
  - Réalisation d'un front ensuivant la maquette (avec respect des codes couleur de l'entreprise)
  - Regrouppement dans un fichier .js unique, pour une plus grande facilité d'intégration
  - Creation de variables pour gerer la validité de la session et mises dans le fichier « .env » (MAX_SESSION_DURATION ; MAX_CHARACTERS)
  - Ajout d'un système de log error

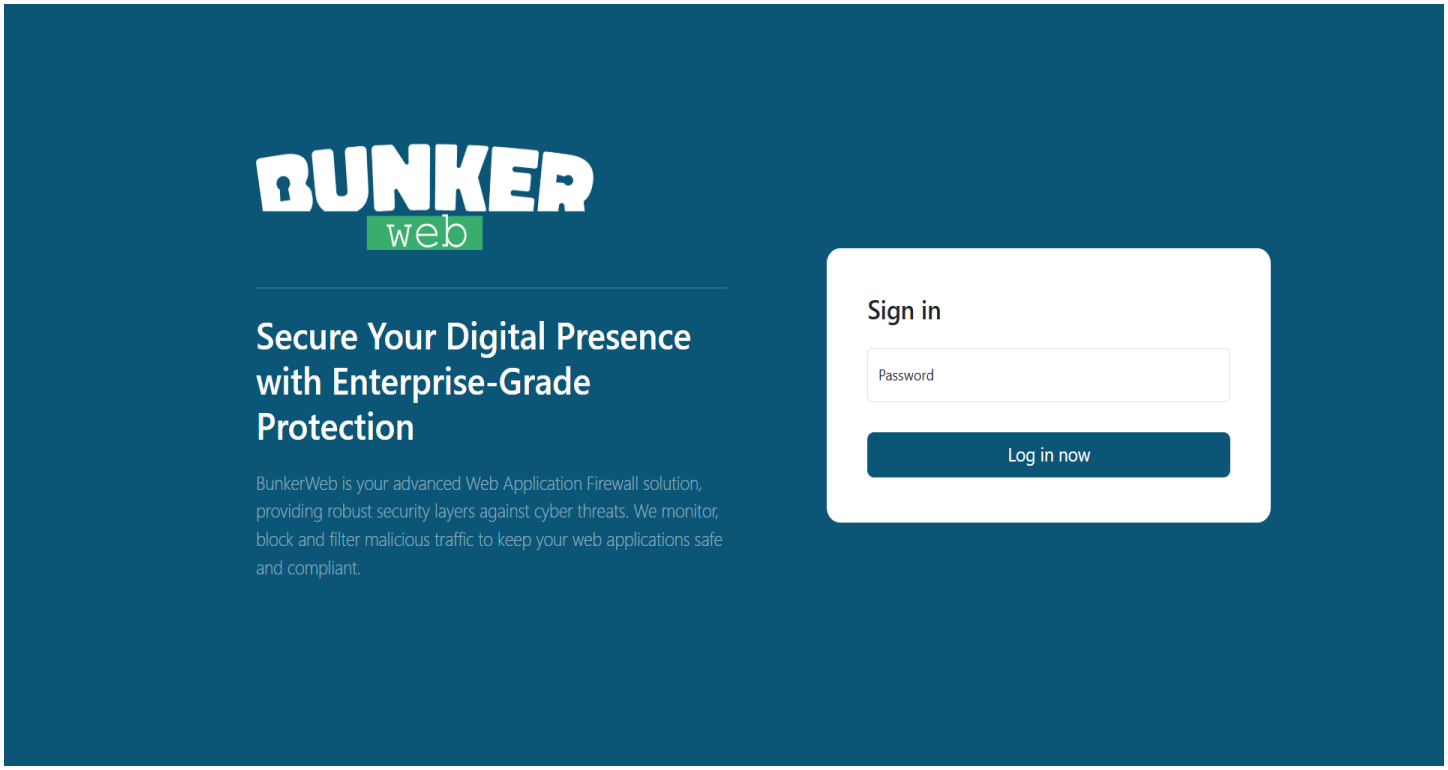
- Jalon 6 – Interface utilisateur conforme à la maquette
    - L'UI respecte les specifications demandées
    - Les sessions sont gérées
    - Les erreures s'affichent bien

Phase 7 – Beta test et finalisation

- Objectif :
    - Pouvoir tester le chatbot en conditions réelles
    - Corriger d'eventuels bugs et apporter des am éliorations
    - Ajouter un système d'authentification avec password

- Taches :
    - Création d'une page de test avec acces via mot de passe
    - Test du chatbot (UI et fonctionnement)
    - Récolte des retours utilisateurs
    - Correction et ajustements (modification du Markdown pour une meileur visibilité)
    - Ajout d'un login avec password, ajout d'undécorateur sur les routes pour plus de sécurité

- Jalon 7 – Version stable livrée
    - Le chatbot est testé, complet et fonctionnel
    - Authentification intégrée pour la beta et protection des routes
    - Le projet est prêt à etre intégré



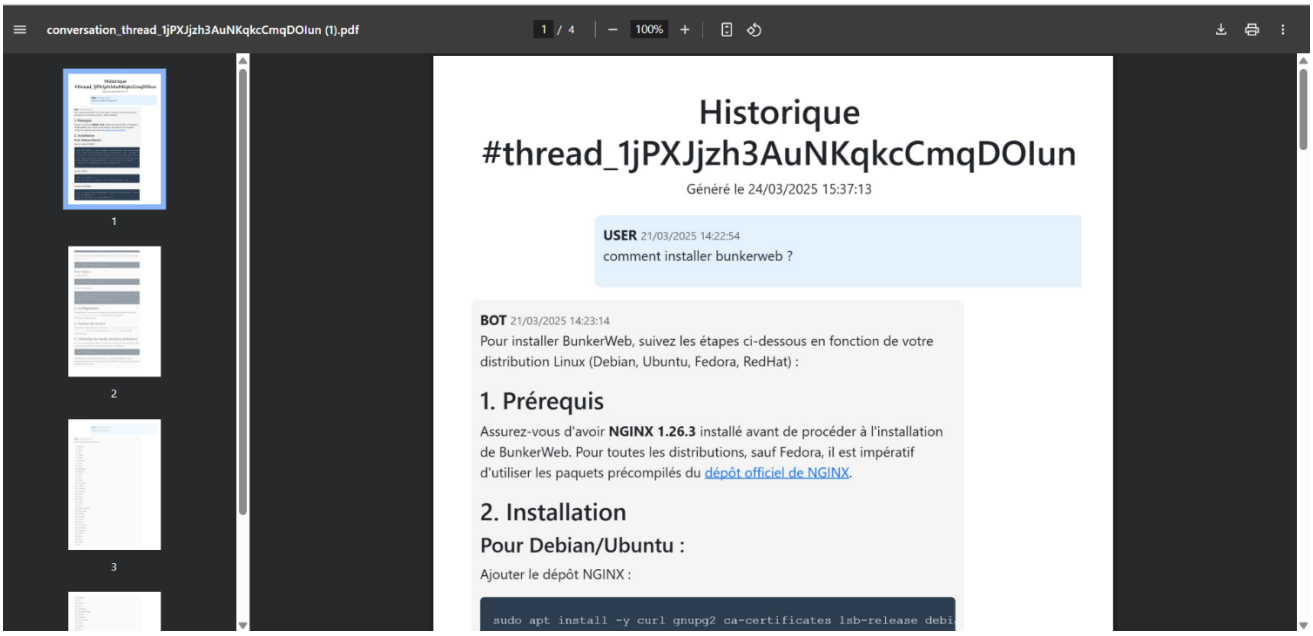
Interface mise en place pour la beta test

Phase 8 – Ameliorations

- Objectif :
  - Ajout de fonctionnalitées demandées par le superieur
  - Amelioration du produit existant

- Taches :
  - Ajout d'un système d'export des threads de conversation sous forme de PDF coté admin
  - Ajout d'un entrypoint.sh afin de mieux gerer la mise a jour de la base de données et ses migrations
  - Ajout d'un bouton de copie sur les parties de code fourni par l'IA

- Jalon 8 – Amelioration continue
  - Possibilité d'exporter des conversations
  - Base de données mieux gérée



Pdf telechargeable avec visuel amélioré

Phase 9 – Documentation

- Objectif :

- Fournir une documentation technique du projet
- Proposer un guide clair pour les développeurs et les utilisateurs administrateurs (login)

- Taches :
  - Rédaction d'un README.md
  - Mise a jour du fichier tout au long du développement
  - Intégration des informations techniques, environnementales et structurelles

- Jalon 9 – Fichier README
  - Livraison du fichier README.md validé comme documentation technique du projet, validé par le superieur

VIII Specifications techniques

Ce projet repose sur des technologies adaptées à l'entreprise. En effet, l'équippe de l'entreprise (start-up) ayant déjà des technologies de predilection, il a falu s'adapter à ces dites technologies afin de rester cohérent avec le fonctionnement de l'entreprise.

Le projet repose en somme sur une architecture solide et securisée, avec des logiqes séparées (front/back, base de données, interface de gestion de l'IA coté Azure). Le projet peut donc etre facilement mmodifiable et scalable.

Sont présenté ici les technologies utilisés et l'organisation choisie dans le cadre de ce projet :

Environnement de travail

Technologies utilisées :

Outil / Language	Version	Role
Python	3.11	Language principale pour le backend
Flask	3.1	Framework léger pour le developpement d'API RESTfull
MySQL	X	Base de données relationnelle pour stocker les conversations et autres données

Docker	X	Conteneurisation des services (Flask, MySQL)
pytest	8.3.5	Framework de tests pour valider le bon fonctionnement des routes et des fonctionnalités
Python-dotenv	1.0.1	Gestion des variables d'environnement (clés API, URLs)
Flask-Migrate	4.1	Gestion des migrations de la base de données

Front-End

Outils et bibliothèques utilisés :

Outil / language	Role	Impact sur le developpement
HTML / CSS	Structure et design	Création de l'interface utilisateur et admin
Bootstrap 5	Framework CSS	Utilisation de composants prêts à l'emploi pour accélérer le développement du design
DataTables.js	Tableaux interactifs	Mise en place de tables dynamiques et interactives dans le tableau de bord admin
JavaScript	Dynamique côté client	Gestion des soumissions de formulaire, appels API, et mise à jour des éléments de la page sans rechargement
DOMPurify	Sécurisation HTML	Purification du contenu dynamique pour éviter les risques de XSS (cross-site scripting)
Marked.js	Conversion Markdown	Conversion du contenu Markdown en HTML pour intégrer les réponses du chatbot de manière sécurisée
html2pdf.js	Génération de PDF	Export des résultats ou des rapports en format PDF depuis le front-end

Back-End

Outils et bibliothèques utilisés :

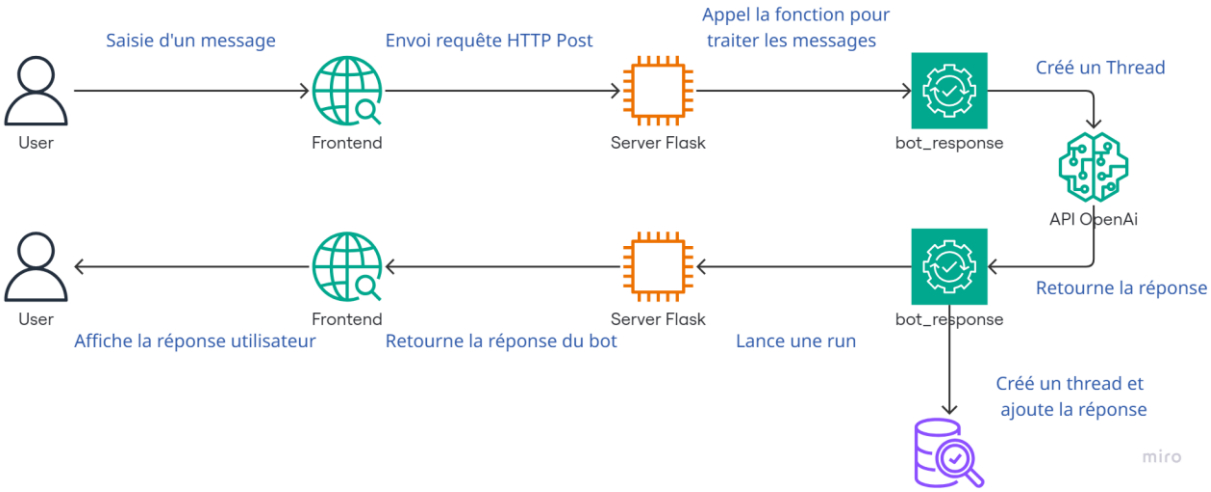
Outil / Langage	Role	Impact sur le développement
Flask 3.1.0	Framework web	Framework léger pour développer des API RESTful
Flask-SQLAlchemy 3.1.1	ORM	Facilite l'interaction avec la base de données MySQL
Flask-Migrate 4.1.0	Migration	Permet de gérer facilement les évolutions du schéma de la base de données
Flask-Cors 5.0.0	Cross-Origin	Permet de gérer les appels API sécurisés entre le front-end et le back-end
Flask-BasicAuth 0.2.0	Authentification	Sécurisation des routes via une authentification basique
Flask-WTF 1.2.2	Formulaires	Sécurisation et gestion des formulaires sur le back-end
PyMySQL 1.1.1	Client MySQL	Connexion et gestion des données dans la base MySQL
OpenAI Python SDK 1.60.0	API ChatGPT	Intégration de l'API ChatGPT via Azure pour alimenter le chatbot
python-dotenv 1.0.1	Variable d'environnement	Sécurisation des données sensibles (ex. clés API, URL de base de données)
cryptography 44.0.0	Securisation	Gestion de la sécurité des données via cryptographie
pytest 8.3.5	Tests unitaires	Mise en place de tests pour valider les fonctionnalités du back-end
Markdown 3.7	Conversion Markdown	Conversion des contenus Markdown envoyés et reçus du chatbot

On constate donc que les logiques front/ back ont bien été séparées. L'utilisation de technologies legeres respecte bien les consihnes de l'entreprise, et sont sufisantes et efficaces pour le projet.

Architecture logiciel

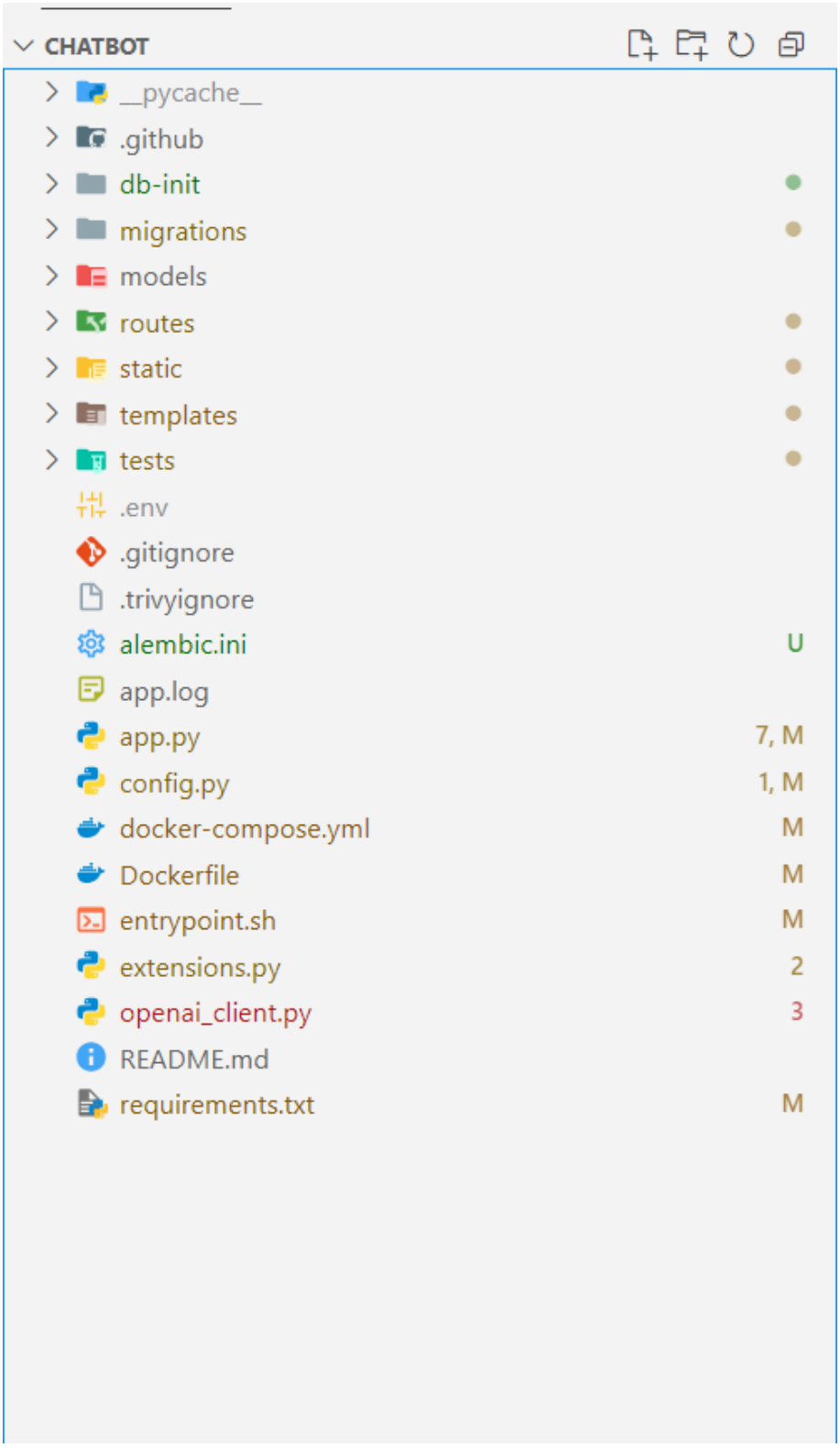
Le projet repose sur une architecture client / server classique, qui pivote autour de Flask coté back-end, et d'une interface web coté front-end. Cette structure permet à la fois une gestion claire du projet avec separation des logiques, mais aussi une scalabilité possible.

L'application suit une logique de communication simple entre le client et serveur. Lorsque qu'un utilisateur soumet une requete au chatbot, celle-ci est transmise au server Flask pour le traitement, qui à son tour le transmet à l'API OpenAI pour l'obtention d'une réponse. Les echanges sont enregistré dans une base de données via Flask pendant cette opération. Elles sont par la suite accessibles via l'interface d'administration.



Arborescence technique

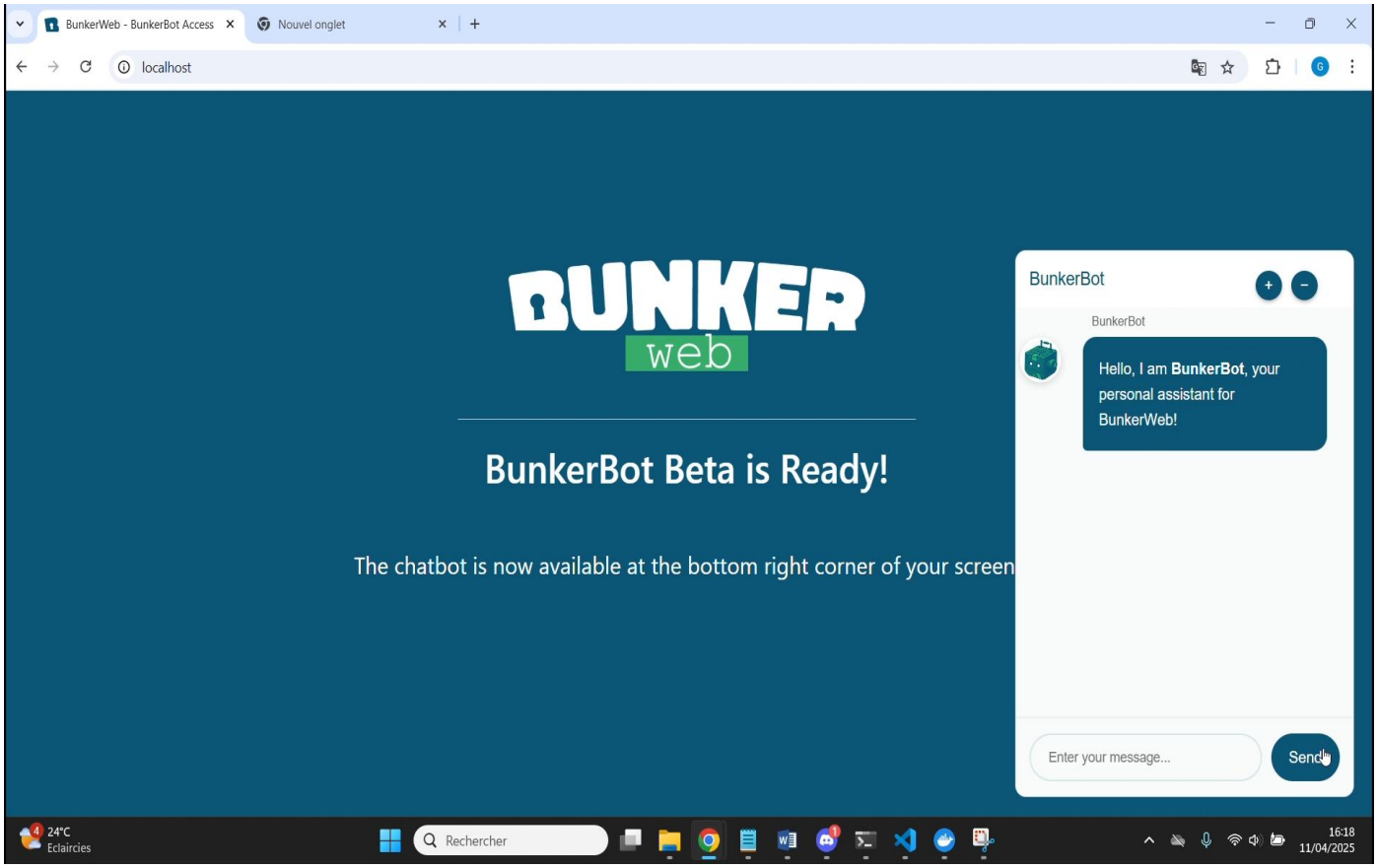
L'application est structurée avec pour objectif de facilité la lisibilité du projet et prmettre une meilleure maintenabilité. La logique des routes utilisé pour le fonctionnement du chatbot est séparée de celle utilisé pour le fonctionnement de la page admin. Les composants visuel (HTML, CSS, JS) sont regroupés, tout comme le model de données. Les fichiers de configuration sont facilement identifiables à la racine du projet.



Arborescence technique du projet

Interface utilisateur avec le chatbot

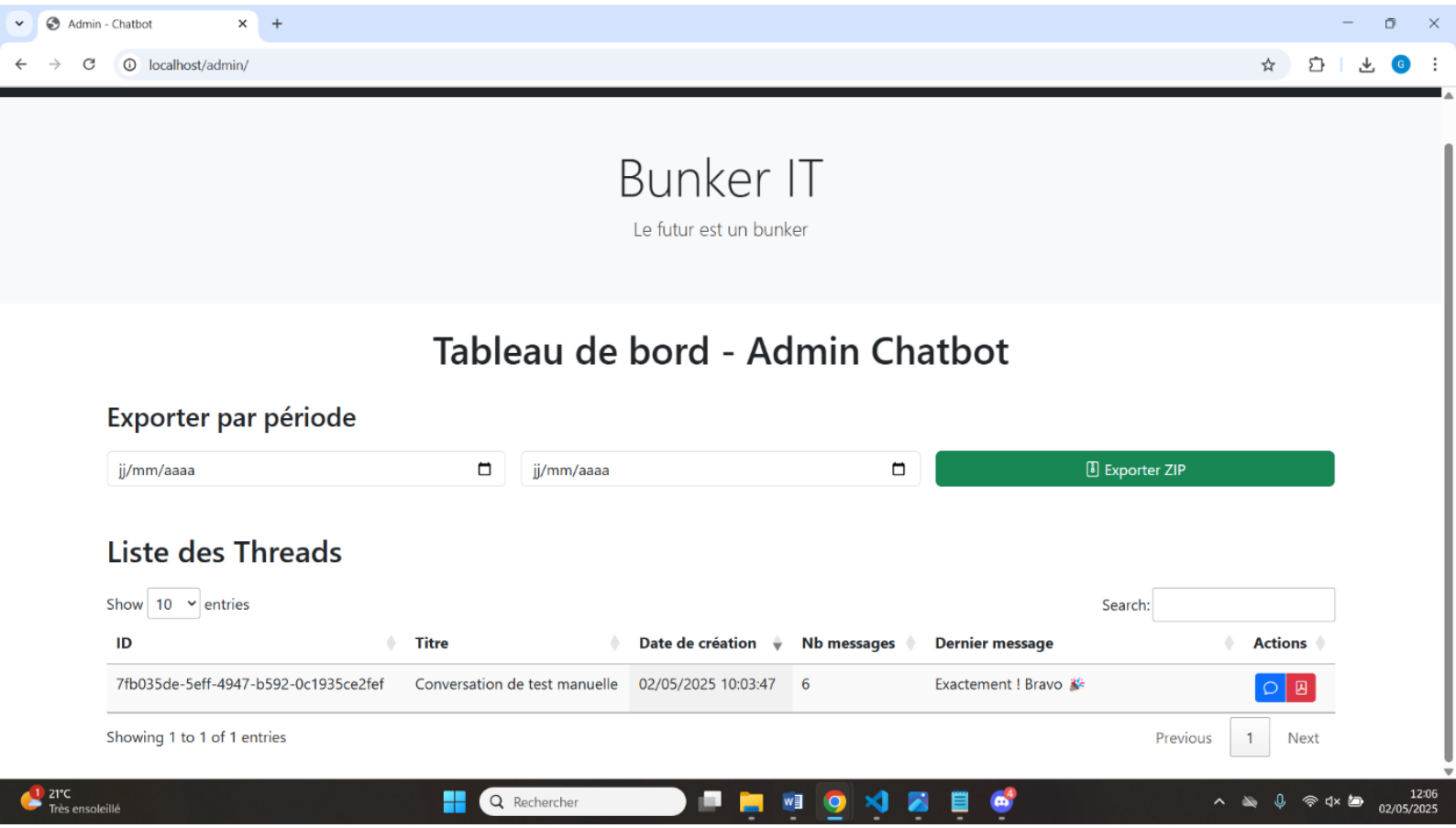
L'interface utilisateur à évolué au fil de la construction du projet. La page principale permet à un utilisateur de communiquer avec le chatbot. Les messages s'envoient de manière fluide et asynchrone avec JavaScrit, afin d'éviter des rechargements de page.



Interface utilisateur

Interface d'administration via la page admin

L'interface admin permet l'affichage des informatrions d'utilisation du chatbot, notemment les threads et les messages de conversations, ainsi que des informations globales relatives à la frequence d'utilisation du chatbot. Les conversations sont organisées grace à Datatables ; on a ainsi une pagination, un tri automatique des colonnes par date, nombre de messages… Cette interface permet le suivi des utilisations de l'utilisateur.



Interface admin

IX Développement

Le développement s'est déroulé en plusieurs phases, en passant de la gestion des routes via le backend, à l'intégration de la documentation du produit de l'entreprise en index via le portail Azure. Nous allons détailler les differentes etapes techniques dans ce chapitre.

Developpement front-end :

Le front-end à une contrainte majeur demandé par le supérieur : tout le code doit etre dans une page pour une intégration facile. Cela a demandé une mise en place spécifique. Plusieurs axes ont été exploré lors de la realisation de ce code ; l'interface utilisateur, une interaction souple et asyncrone, et un aspect de sécurité. Voyons ces differents points :

1) Intégration du DOM dynamique

L'interface du chatbot est entierement générée et injecté dynamiquement dans le DOM à l'aide de JavaScript :

- Création d'un container #chatbot-container contenant l'en-tête, le corps de conversation et le pied avec le champ de saisie.

```
const style = document.createElement('style');
      style.innerHTML = `
        #chatbot-container {
```

- Boutons d'extension (+) et de fermeture (−) pour une UX flexible

2) Interaction et logique utilisateur

L'UI révele plusieurs aspects techniques avec pour objectif le confort de l'utilisateur :

- L'envoi de mesages se fait via un champ de texte. Le bouton Send declanche une requete POST vers /chat

```
// Event handler click
sendBtn.addEventListener('click', () => {
    if (isWaitingForResponse) return;
    const userMessage = input.value.trim();
    if (userMessage) {
        sendMessage(userMessage); // <-- Déclenche l'envoi
    }
});

// Event handler "enter"
input.addEventListener('keydown', (event) => {
    if (event.key === 'Enter' && !event.shiftKey) {
        event.preventDefault();
        if (isWaitingForResponse) return;
        const userMessage = input.value.trim();
        if (userMessage) {
            sendMessage(userMessage); // <-- Déclenche l'envoi
        }
    }
});

    // Function for POST request
async function sendMessage(message) {
    // [...] (préparation des données)
    const response = await fetch(API_CHAT, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ message, thread_id: threadId }) // <--
Corps de la requête
    });
    // [...] (gestion de la réponse)
}
```

- Scroll automatique vers le bas à chaque nouveau message pour maintenir le fil de conversation visible
- Gestion de l'affichage dynamique des messages avec distinction bot / user

3) Comportement adaptatif de l'UI

Pour une experience plus agréable, l'UI inclut :

- Mode minimized (réduit) automatiquement à l'ouverture, avec retour à la taille par défaut ou enlarged (agrandie) selon les actions utilisateur.
- Responsiveness géré via CSS et événements (window.load, resize, click, etc.).

- Style CSS injecté dynamiquement avec mise en forme personnalisée, ombres, bordures arrondies et animation fluide.

4) Sécurité client et affichage Markdown

- Utilisation de la librairie marked avec configuration personnalisée (breaks, gfm, etc.) pour un rendu propre du Markdown côté client.
- Les messages Markdown sont sanitisés via DOMPurify avant affichage pour éviter toute injection de code malveillant.

```
// Fonction displayMessage avec parsing Markdown et sanitization
function displayMessage(content, sender) {
    const cleanedContent = content.replace(/【.*?】/g, '');  //
Nettoyage préalable
    const messageDiv = document.createElement('div');
    messageDiv.classList.add('message', sender);

    if(sender === 'bot') {
        // Parsing Markdown + Sanitization DOM
        messageDiv.innerHTML = `
            <div class="bot-content">
            ${DOMPurify.sanitize(marked.parse(cleanedContent))} <!--
Double sécurité -->
            </div>
        `;
    }
    // [...] (reste du code)
}
```

5) Ergonomie et accessibilité

- Affichage des messages du bot avec avatar personalisé de l'entreprise (bunky.png) et nom officiel (Bunkerbot)

- Ajout d'un bouton donant la possibilité de copier le code généré par l'IA pour améliorer l'experience utilisateur.

```
// Ajout du bouton de copie sur chaque bloc <pre>
if (sender === 'bot') {
    const preElements = messageDiv.querySelectorAll('pre');
    preElements.forEach(pre => {
        const copyButton = document.createElement('button');
        copyButton.className = 'copy-code-btn';
        copyButton.setAttribute('aria-label', 'Copy to clipboard');
        copyButton.setAttribute('data-clipboard-target', `#${pre.id} >
code`);

        pre.appendChild(copyButton); // <-- Ajout du bouton

        copyButton.addEventListener('click', (e) => {
            const codeElement = pre.querySelector('code');
            // [...] Logique de copie
        });
    });
}
```

- Comportement non bloquant : le bouton Send est désactivé pendant le temps de réponse de l'IA (avec isWaitingForResponse), afin de proteger d'un spam de requetes et de garder un ordre de conversation utilisateur/bot pour des threads plus coherents.

6) Connexion avec le backend

Les appels API sont synchronisés avec les endpoints Flask :

- POST /start_chat au démarrage
- POST /chat pour chaque message utilisateur
- GET /session_status pour verifier la validité de la session en cours (permet de valider l'envoi du message utilisateur ou l'informer si la session n'est plus valide)

Le développement backend :

Le cœur du projet repose sur l'utilisation de Flask pour gerer les routes de l'application. On s'articule notamment particulierement sur :

- Une route principale pour la reception des requetes utilisateur depuis l'interface
- Une logique de traitement pour l'appel à l'API OpenAi
- Un retour structuré des reponses de l'IA, pretes à etre affiché coté front

Le code gere egalement la structure des requetes API, et permet de traiter les mesages et la conversion du resultat de l'IA en format exploitable. On a séparé les routes de la page admin dans un fichier séparé, afin de plus facilement diferencier les logiques.

1) Routes gérées via Flask

```
@app.route('/start_chat', methods=['POST'])@app.route('/start_chat',
methods=['POST'])
```

La route POST /start_chat à pour role de créer un nouveau thread avec le thread_id généré par le chatbot, coté Azure.

```
@app.route('/chat', methods=['POST'])
```

La route POST /chat a pour role d'envoyer un message utilisateur à l'API Azure, puis de récupérer une réponse de l'IA. Il sert en outre à persister les échanges en base de données.

```
@app.route('/thread/<string:thread_id>/messages', methods=['GET'])
```

La route GET /thread/<thread_id>/messages a pour objectif de renvoyer l'historique des messages d'un thread donné. Il permet à l'utilisateur de ne pas perdre sa conversation en cas de rafraichissement de la page.

```
@app.route('/session_status', methods=['GET'])
```

La route GET /session_status permet d'obtenir le temps et le nombre de characteres restant pour la validité de la session.

2) Communication avec l'IA

Afin de comuniquer avec l'IA, les routes utilisé comme vu precedement, une partie de developpement specifique à l'utilisation d'une API utilisant un model de language à été réalisé.

Au dela de la partie de développement du projet, l'utilisation de l'IA demande une connaissance particulière de l'interface de parametrage de cet outil. Ce parametrage peut dépendre en outre de l'entreprise fournissant ce service. Dans le cas present, viale service Azure OpenAI, c'est par le portail Azure foundry que se fait une partie de la mise en place de l'IA.

Dans un fichié séparé nommé openai_client.py, plusieurs actions sont menées :

```
client = AzureOpenAI(
    azure_endpoint=os.getenv("AZURE_OPENAI_ENDPOINT"),
    api_key=os.getenv("AZURE_OPENAI_API_KEY"),
    api_version="2024-05-01-preview"
)

assistant_id = os.getenv("AZURE_OPENAI_ASSISTANT_ID")
```

Dans un premier temps, une identification est requise afin d'acceder à notre modele. Les variables, sensibles sont stockées dans un fichier .env.

```python
def create_new_thread():
    try:
        thread = client.beta.threads.create()
        return str(thread.id)
    except Exception as e:
        print(f"Erreur création thread : {str(e)}")
        return None
```

Cette fonction a pour but de créer un nouveau thread coté Azure, et de récupérer le thread_id ainsi généré. On gere egalement une erreur possible.

```python
def bot_response(user_message, thread_id):
    try:
        # Adding user message
        client.beta.threads.messages.create(
            thread_id=thread_id,
            role="user",
            content=user_message[:5000]
        )

        # Run with timeout creation
        run = client.beta.threads.runs.create(
            thread_id=thread_id,
            assistant_id=assistant_id,
            timeout=30
        )

        # Checking with exponential backoff
        start_time = time.time()
        while True:
            run_status = client.beta.threads.runs.retrieve(
                thread_id=thread_id,
                run_id=run.id,
                timeout=10
            )

            if run_status.status == "completed":
                break

            if run_status.status in ["failed", "cancelled", "expired"]:
                return f"Erreur OpenAI: {run_status.last_error}"

            if time.time() - start_time > 30:
                return "Erreur: Timeout de l'API OpenAI"

            time.sleep(1)

        # Recieving answer
        messages = client.beta.threads.messages.list(
            thread_id=thread_id,
            timeout=10
        )
        return messages.data[0].content[0].text.value

    except Exception as e:
        return f"Erreur: {str(e)}"
```

Dans cette fonction, plusieurs objectifs sont remplis :

- Ajout d'un message utilisateur dans une conversation
- Lancement d'un run pour obtenir une reponse. Ce run va boucler toutes les secondes pour verifier si l'on a une reponse de l'IA, jusqu'à un maximum de 30 secondes.
- On récupère la réponse générée par l'IA
- Système de capture d'erreur

3) Validation des données :

Pour plus de solidité, les données sont controllées sous plusieurs aspects :

```
# JSON format checking
        if not request.is_json:
            logger.error("Requête reçue avec un format non JSON")
            return jsonify({"error": "Internal server error"}), 500
```

On contrôle que le format est bien au JSON

```
# Message lenght cheking
        if not isinstance(user_message, str) or
len(user_message.strip()) == 0:
            return jsonify({"error": "Empty message"}), 400
        elif len(user_message) > MAX_MESSAGE_CHARACTERS:
            logger.error(f"Message too long ({len(user_message)}
caracteres)")
            return jsonify({
                "error": f"Message too long
({len(user_message)}/{MAX_MESSAGE_CHARACTERS} caracteres)"
            }), 400
```

Ici, on verifie que le message ne dépasse pas le nombre de characteres maximum. La variable est facilement modifiable dans le fichier .env.

```
# [4] Timout expiration checking
        session_age = (datetime.utcnow() -
thread.created_at).total_seconds()
        if session_age > MAX_SESSION_DURATION:
            logger.info(f"Thread expired (time) {thread_id}
({session_age}s)")
            thread.status = 'expired'
            db.session.commit()
            return jsonify({"error": "Session expired"}), 403
```

Vérification de la validité des threads via la variable MAX_SESSION_DURATION et leurs expirations.

4) Implementation de l'index documentaire

Le model de la base de données est gérée dans un fichier models.py (séparation des logiques)

- La base de données contient 2 tables :
  - Thread ; qui represente la conversation avec l'IA
  - Message ; qui correspond aux messages de l'utilisateur.

```
from extensions import db
from datetime import datetime

class Thread(db.Model):
    __tablename__ = 'threads'
    id = db.Column(db.String(36), primary_key=True)
    title = db.Column(db.String(255), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    status = db.Column(db.String(20), default='active')
    messages = db.relationship('Message', backref='thread', lazy=True)

class Message(db.Model):
    __tablename__ = 'messages'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    thread_id = db.Column(db.String(36), db.ForeignKey('threads.id'),
nullable=False)
    content = db.Column(db.Text, nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    origin = db.Column(db.String(10), nullable=False)
```

- Le système fonctionne comme un index conversationnel :
  - Chaque message est lié à un thread avec thread_id
  - Les mesages sont clasé chronologiquement avec « created_at.asc() »

```
messages =
Message.query.filter_by(thread_id=thread_id).order_by(Message.created_a
t.asc()).all()
```

5) Connexion MySQL et persistence des données

La connexion à la base de données MySQL est gérée grace à SQLAlchemy et PyMySQL. Les données sensibles de connexion à la base de données sont swtocké dans le fichier .env .

```
# Configuration of database
class Config:
    SQLALCHEMY_DATABASE_URI = (
        f"mysql+pymysql://{os.getenv('MYSQL_USER')}:
{os.getenv('MYSQL_PASSWORD')}@"
        f"{os.getenv('MYSQL_HOST')}:
{os.getenv('MYSQL_PORT')}/{os.getenv('MYSQL_DB')}"
    )
```

Les echanges sont stockés dans la base de données :

- Lors d'un chat, un objet Message est créé et ajouté en base
- Les threads et leurs metadonnées (date de création, origine du message utilisateur/bot) sont persités

Les extentions sont initialisés dans un fichier extensions.py ; avec db = SQLAlchemy() et migrate = Migrate() pour la gestion des migrations (flask db migrate, flask db upgrade).

6) Gestion des erreurs

Afin de donenr plus de resilience à notre projet et de mieux pouvoir identifier d'eventuels origines d'erreures, un système de gesion des erreurs (try/except) et de journalisation (logger) à été mis en place.

Lors de la connexion MySQL au démérage, un test explicite de connexion est réalisé avec db.engine.connect() .

Divers cas d'erreurs ont été prévu :

- Thread introuvable : renvoi un code erreur 404 avec jsonify({"error": "Invalide session"})
- Expiration de session : code 403
- Message trop long ou vide : code 400 avec messages explicites

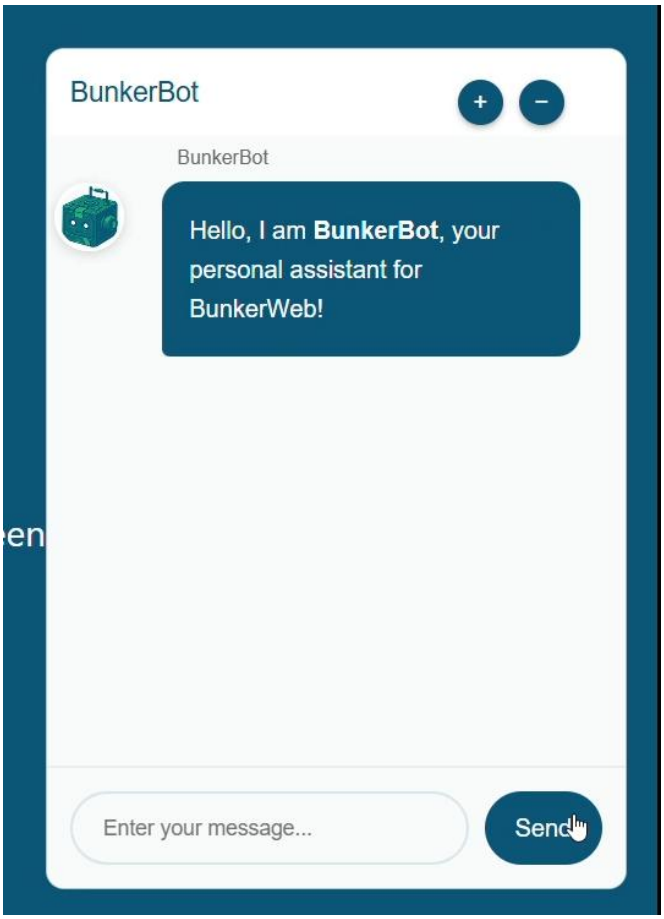
Un système de journalisation avancé est installé :

- Logs en production dans un fichier app.log avec RotatingFileHandler
- Niveau de log configurable dans le fichier .env
- Journalisation utilisée dans tout le code

```
logger.error(f"Login error: {str(e)}", exc_info=True)
logger.warning("Session status request without thread_id")
logger.debug(f"Total thread caracteres {thread_id}: {total_chars}")
```

Intégration front-end

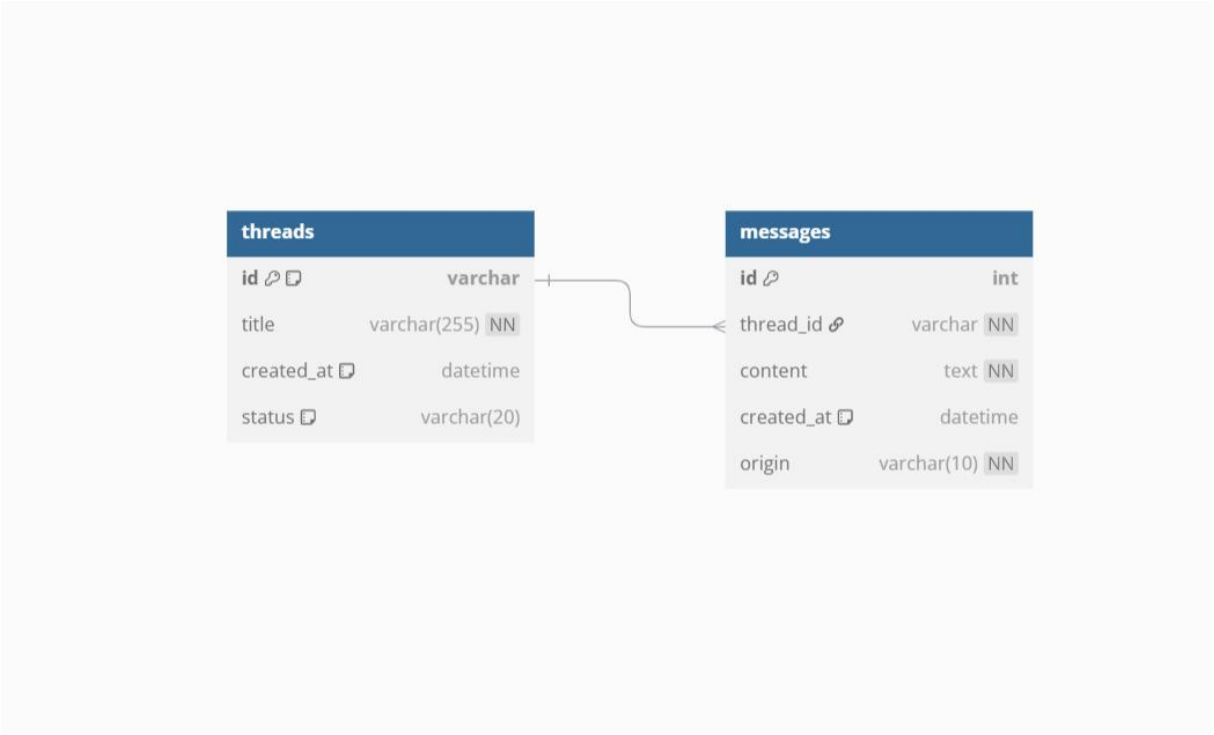
L'objectif principale et la condition demandée était d'integrer le front entier dans un seul fichier JavaScript afin de pouvoir l'intégrer de manière facile dans une page existante. Plusieurs defis ont été relevé afin de rendre l'utilisation plus agréable pour l'utilisateur, ajout de bouton de copie par exemple. Bootstrap a été utilisé pour sa simplicité d'intégration et d'utilisation.



Interface finale du chatbot

Base de données persistante

La base de données MySQL enregistre toutes les conversations utilisateur, afin d'etre disponible pour une consultation et exploitation ulterieur



MCD de la base de données

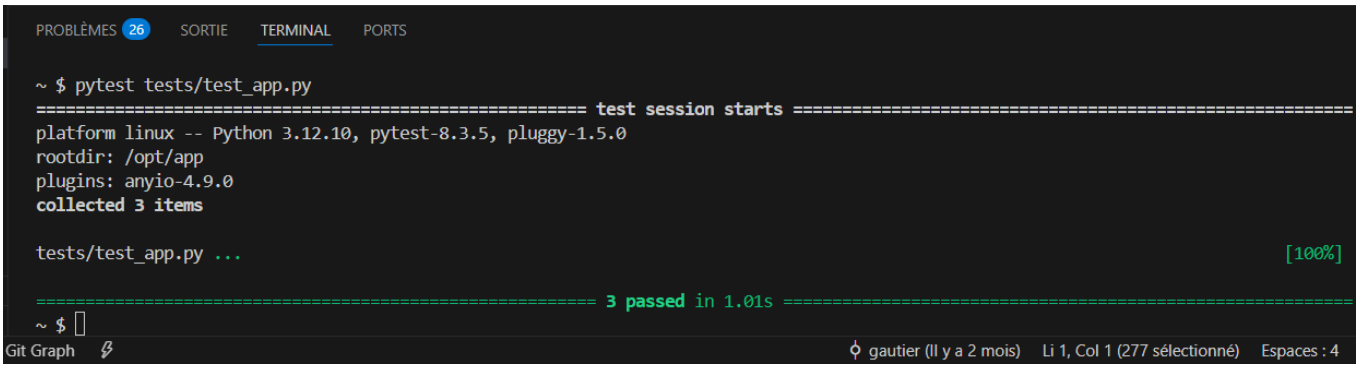
X Tests

Tests unitaires

Malgres un imperatif de temps reduit, des tests unitaires ont été réalisés pour vérifier la logique de connexion de l'administrateur. Ces tests sont essentiels pour s'assurer du bon fonctionnement de la connexion administrateur. Afin de pouvoir réaliser ces tests, pytest à été choisi comme outil permettant de les executer.

```python
import pytest
import sys
import os
from dotenv import load_dotenv
env_path = os.path.join(os.path.dirname(__file__), '..', '.env')
load_dotenv(env_path)
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from app import app as flask_app

@pytest.fixture
def client():
    flask_app.config['TESTING'] = True
    with flask_app.test_client() as client:
        with flask_app.app_context():
            yield client

def test_login_success(client):
    correct_password = os.getenv('BETA_PASSWORD', 'ilovebeer1234')

    response = client.post('/login',
                           data={'password': correct_password},
                           follow_redirects=True)

    assert response.status_code == 200
    assert b'Session expir' not in response.data
    with client.session_transaction() as sess:
        assert sess.get('authenticated') is True

def test_login_wrong_password(client):
    response = client.post('/login', data={'password':
'wrongpassword'}, follow_redirects=True)
    assert response.status_code == 200
    assert b'Unauthorized access' in response.data
    with client.session_transaction() as sess:
        assert sess.get('authenticated') is not True

def test_login_no_password(client):
    response = client.post('/login', data={}, follow_redirects=True)
    assert response.status_code == 200
    assert b'Unauthorized access' in response.data
```

Tests unitaires réalisés



Résultats obtenus

Tests manuels

Tout au long de la production du chatbot, des tests manuels ont été effectués afin de verifier que le chatbot fonctionnait bien en local dans un premier temps. Ces tests consistaient en :

- L’envoie de messages via l'interface utilisateur et reception de la reponse (message généré par IA, test de contexte…)
- Test de fonctionnement des fonctionalitées ; bouton copier, bouton pour agrandire/reduire, blocage du champ de saisie en attente de la réponse…)
- Verification de la persistence en base de données et affichage dans l’interface administrateur

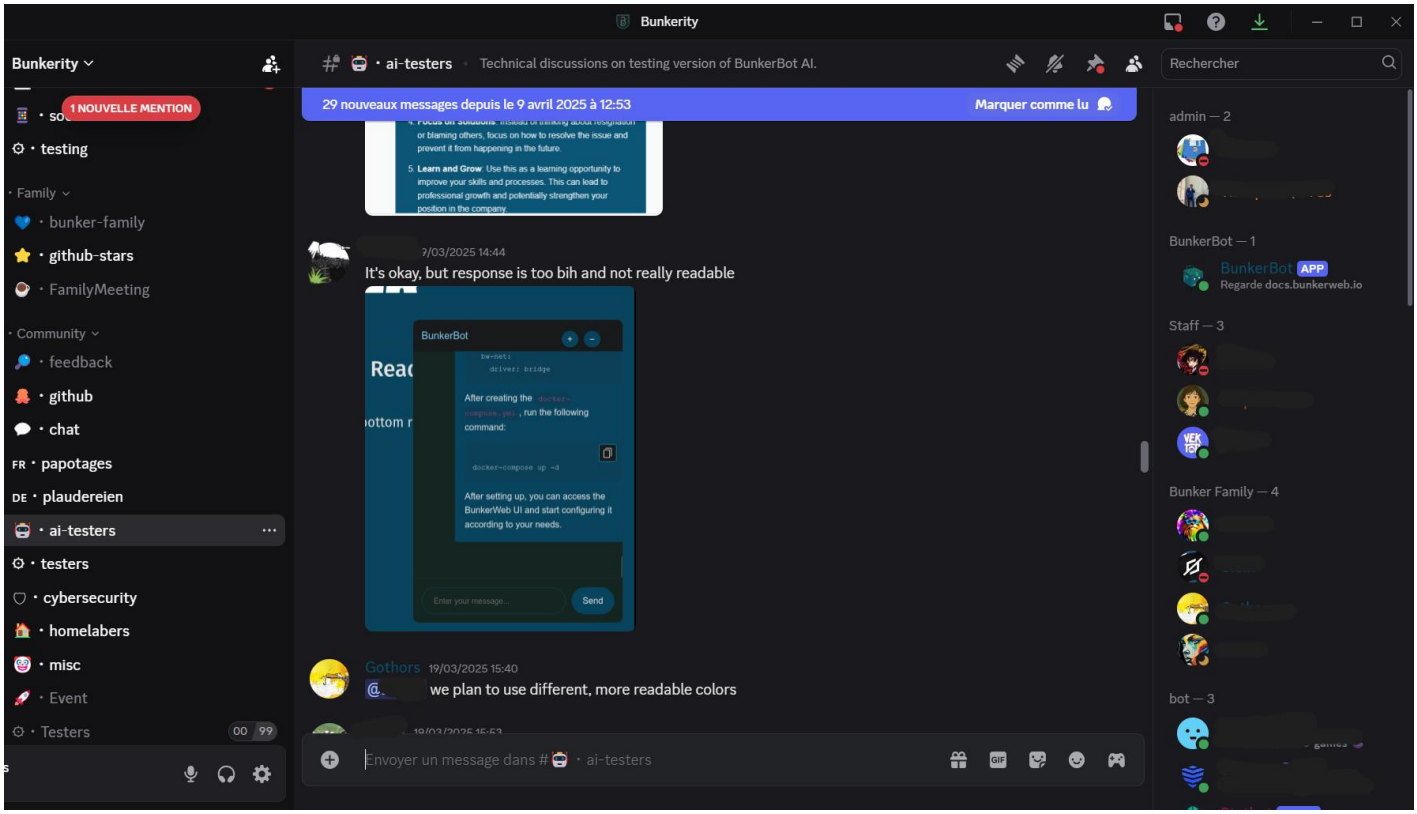
1) Intégration continue

Le projet a été intégré dans un pipeline d'intégration continue utilisant GitHub Actions. À chaque push sur le dépôt GitHub, des tests automatisés étaient déclenchés pour vérifier la stabilité du code. L'application étant conteneurisée avec Docker, les workflows GitHub Actions construisaient l'image Docker et exécutaient les tests dans un environnement isolé. L'utilisation d'un fichier .env permettait de gérer les variables d'environnement, facilitant ainsi le déploiement dans différents environnements (développement, test, production).

2) Test en production (version beta)

Une version beta du chatbot, accessible via mot de passe, à été mis à la disposition de la comunauté discord des utilisateurs du produit de l'entreprise, cet environnement en conditions reels avec des developpeurs était une oportunité excellente pour avoir des retours professionnels sur le chatbot.

Dans ce contexte, les messages utilisateurs ont été analyser afin d'ameliorer le produit et de corriger d'eventuels bugs. Cette approche a contribué à l'amélioration du produit (Exemple ; changement de certaines couleurs de marked afin que les reponses du bot et notament ses citations soient plus lisibles).



Retour utilisateur de la beta-test

XI Resultats obtenus

Fonctionnalités terminées

A travers le developpement du chatbot, l'apport d'une maquette, et l'analyse des retours utilisateurs, plusieurs fonctionnalitées ont été réalisées :

- Interface utilisateur avec code couleur et images de l'entreprise, fonctionalitées de forme de pastille discrete mais visible et d'agrandissement, possibilité de copier le code cité directement via un bouton

- Base de données permetant la persistance des conversations

- Interface administrateur avec affichage des données (tableau avec les threads et leurs messages, possibilité de venir sur une conversation particuliere, organisation des conversations par date…), traitement des informations d'utilisation, et possibilité d'exportation des threads sous forme de pdf

- Interface du chatbot livrée sous la forme d'un fichier JavaScript unique comme demandé, facilement intégrable

- Mise en place d'un environnement docker et d'un fichier .env

- Mise en place d'un deploiement en beta, avec identificartion via mot de passe uniquement

- Parametrage d'un « assistant », un chatbot personalisé, réalisé via l'interface Azure Foundry

- Enrichissement du bot via l'indexation de la documentation de l'entreprise

Points d'amélioration identifiés

A la fin de la periode de formation, plusieurs points ont été identifiés afin d'ameliorer le produit :

- Amelioration des informations presentées dans l'interface administrateur
- Mise en place d'un bouton d'agrandissement via clické-glissé
- Meilleure gestion, des erreures (difficile d'identifier si certaines erreures viennes du coté Azure ou du chatbot, ameliorer la precision des logs d'erreur)
- Lors d'une recherche sur la documentation de l'entreprise, le chatbot prends l'intégralité de la documentation et l'utilise en token. La documentation n'etant pas grande, la quantitée reste acceptable, mais une optimisation à ce niveau la reduirait les couts d'une fourchette allant de 60% à 90% du cout d'utilisation de token. Solution proposée : passer par un abanement mensuel avec « Azure search », ainsi, ce service ferait la recherche dans la documentation de manière séparé, mais avec un cout fixe, ce qui pourait reduire considérablement les couts.

Réception par l'entreprise

L'entreprise a accueilli favorablement les résultats obtenus :

- Validation des objectifs initiaux : le chatbot répond aux differents besoins de l'entreprise, et est conforma à la maquette présentée
- Statifaction des utilisateurs : les retours de la communauté lors de la beta test ont été globalement positif et encourageant
- Intégration reussi dans l'ecosysteme exixstant : grâce à l'utilisation de Docker et à l'intégration continue via GitHub, le déploiement et la maintenance du chatbot sont facilités. Le projet a passé les tests de l'entreprise avec succes lors du passage en production

XII Competences mobilisées

Ce projet de chatbot basé sur Flask et Azure OpenAI m'a permis de mettre en pratique un large éventail de compétences techniques et organisationnelles, tant sur le plan du développement que sur celui de la gestion d'un environnement de travail professionnel. Voici les principales compétences que j'ai mobilisées au cours de cette réalisation :

Developpement web fullstack

Le projet a necessité des competences à la fois coté backen et cote frontend :

- Backend : Developpement du server Flask, création des routes API, gestion des reqquetes, integration de l'IA
- Frontend : Mise en place d'une premiere interface pour les tests, puis suivi d'une maquette pour une utilisation simple et fonctionnel du chatbot

Consomation d'API tierce (Azure OpenAI)

- Integration et configuration de l'API Azure OpenAi pour générer les reponses du chatbot
- Gestion des clés d'API de manière sécurisé via un fichier .env
- Envoi des requetes, traitements des reponses et adaptation du format pour l'utilisateur

Conception d'interfaces

- Capacité à evaluer les besoins ergonomiques de l'interface du chat
- Mise en place d'un chat agreable visuelement, d'utilisation clair et intuitive
- Prise en compte des retours utilisateurs pour l'amelioration du chatbot

Utilisation de Docker / environnement de développement

- Creation d'un environnement de développement Dockerisé
- Familiarisation avec les fichiers Dockerfile et Dockercompose
- Deploiement dans differents environnement (local, production / beta)

Documentation et travail autonome

- Documentation du projet pour une meilleur comprehension exterieur
- Indication des variables à changer dans le vichier de variables d'environnement
- Adaptation au languages de l'entreprise, auto-formation sur le langage et le contexte de fonctionnement d'un chatbot

XIII Conclusion

Bilan du projet

Ce projet de developpement d'un chatbot intelligent utilisant une API, dans des technologies que je ne connaissais pas (Python, Flask) et dans un environnement de traivail professionnel à été une experience extremement enrichissante.

La création d'un projet en autonomie m'a permis de mettre en pratique les grands conceptes de developpement de manière concrete, et de legitimer mes capacités de developpeur.

Apports techniques et personnels

Ce projet m'a permis de découvrir de nouvelles technologies que je ne connaissais pas, comme Django, un framework web en Python. J'ai appris à créer une application fonctionnelle en m'appuyant sur une maquette pour structurer l'interface. J'ai également conçu une base de données simple et appris à automatiser les migrations pour la production, pour de future modification. J'ai egalement appris à exploiter ces données afin de les afficher dans une interface admin permettant un meilleur visuel sur l'utilisation global du chatbot. Enfin, je me suis initié à l'intégration d'un chatbot, en me renseignant sur le fonctionnement de l'intelligence artificielle et le traitement du langage naturel.

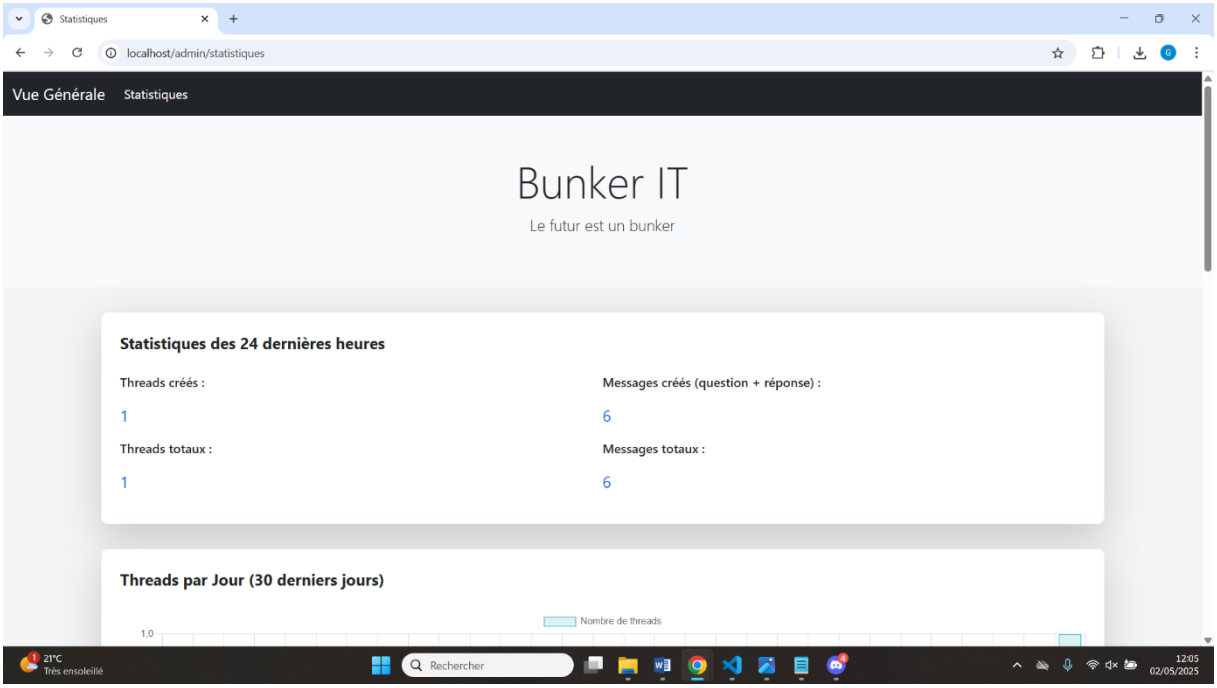
Sur le plan personnel, ce projet m'a appris à travailler de façon autonome. J'ai su organiser mon travail, chercher des solutions par moi-même et m'adapter à des outils ou langages inconnus. Le fait d'avoir évolué en entreprise m'a également permis de mieux comprendre les exigences du monde professionnel, notamment en suivant une maquette de projet et en respectant des objectifs. Cette expérience m'a donné confiance dans ma capacité à apprendre rapidement et à m'adapter à des contextes techniques variés.

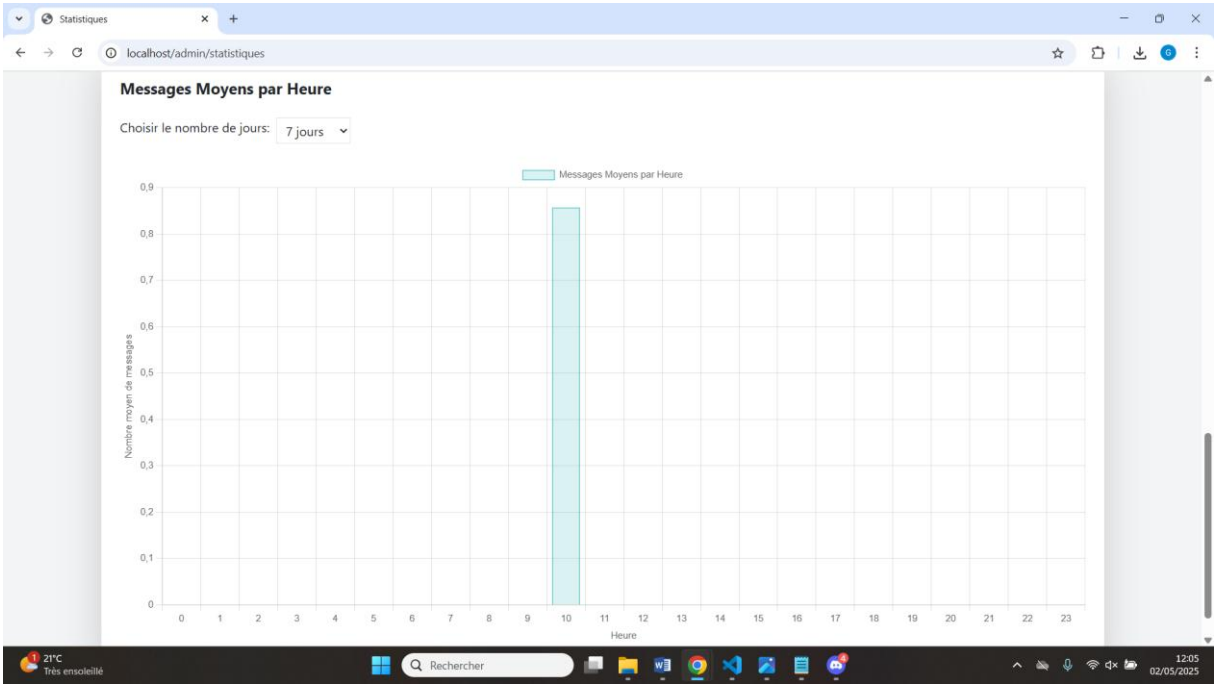
Perspectives d’évolution

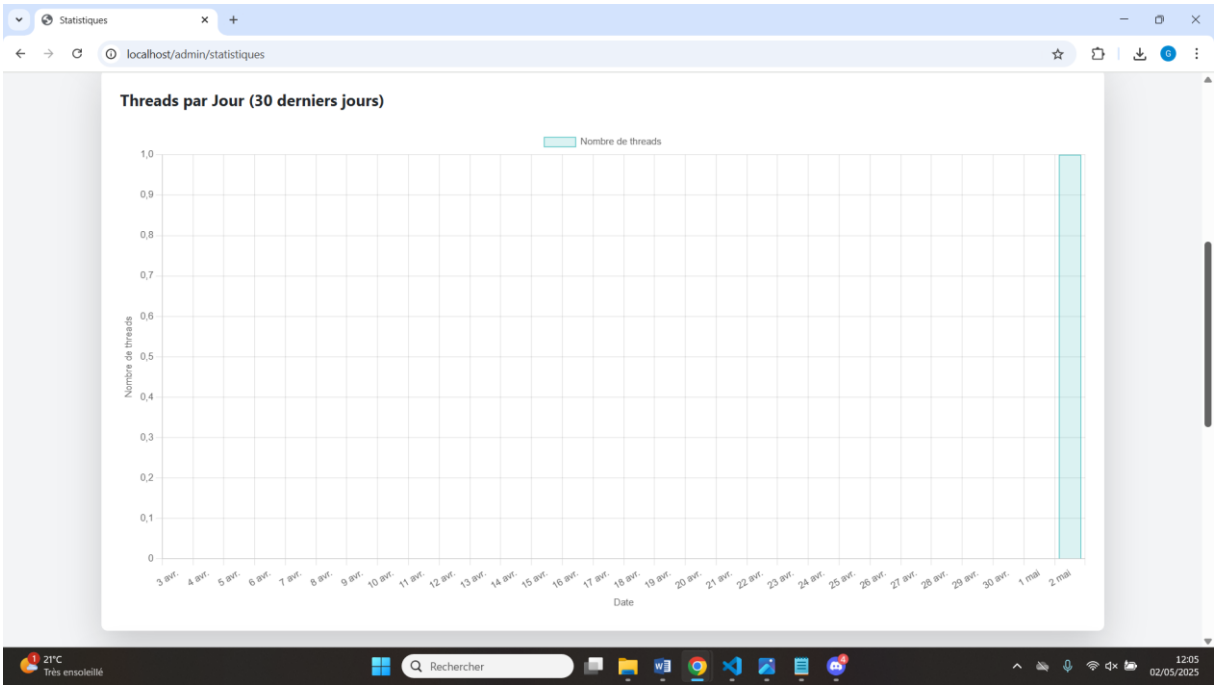
Suite à mon stage en entreprise et à ma formation de manière plus global, j’ai des pistes d evolution sur plusieur aspects. Dans un premier temps, trouver un emploi en tant que developpeur. Ainsi, je pourrais decouvrir les technologies de mon entreprise, et me fixer sur un langage de programmation specifique. Cela me permetrai de developper une expertise sur une technologie ciblée, plutôt que sur un enssemble de languages de manière legere. Cela dit, je reste ouvert au travail en alternance, qui bien qu’il me prive d’un savoir-faire aprofondie, me permettrai de developper ma culture general en tant que developpeur, et ainsi avoir un oeuil plus critique et ouvert sur le fonctionnement des languages et des differents projets, ce qui serait enrichissant.

XII Annexes

Interface admin, coté statistiques :




Statistiques
localhost/admin/statistiques
Messages Moyens par Heure
Choisir le nombre de jours:
7 jours
Messages Moyens par Heure
Nombre moyen de messages
Heure
Rechercher
21°C
Très ensoleillé
12:05
02/05/2025


Statistiques
localhost/admin/statistiques
Threads par Jour (30 derniers jours)
Nombre de threads
Nombre de threads
Date
Rechercher
21°C
Très ensoleillé
12:05
02/05/2025