Rapport Final Détaillé

Legal Assistant

UNIVERSITE INTERNATIONALE DE RABAT

12 janvier 2025

Créé par : amine alami,hamza el bakkali,zine fahd,mohamed el amine rbii

Table of Contents

1. Introduction 2
2. Historique de l'Intelligence Artificielle 2
3. Exigences et Directives du Projet 2
   3.1 Formation de l'equipe 2
   3.2 Portee du Projet 3
   3.3 Technologies Utilisees 3
   3.4 Sessions de Laboratoire (Detaillees) 3
      Session 1 : Conception 3
      Session 2 : Developpement Backend 4
      Session 3 : Developpement Frontend 5
      Session 4 : Integration et Tests 6
4. Gestion de Projet 6
   4.1 Outils de Gestion 6
5. Analyse des Risques 6
   5.1 Risques Techniques 6
   5.2 Risques Organisationnels 6
   5.3 Solutions Apportees 6
6. Plan de Deploiement 7
   6.1 Deploiement Local 7
   6.2 Deploiement sur Heroku 8
7. Analyse Comparative 8
8. Conclusion 8
9. Annexe 9
10. Sources 10

Rapport Final Détaillé

Legal Assistant

1. Introduction

Le projet **Legal Assistant** est une application innovante développée dans le cadre d'un cursus en génie informatique. L'objectif principal est de fournir un outil capable d'apporter des réponses précises à des questions juridiques en utilisant des technologies modernes, notamment l'intelligence artificielle (IA). Cette solution s'adresse à la fois aux professionnels du droit et aux particuliers, avec une ambition de simplifier l'accès aux informations juridiques et de réduire les coûts liés à la recherche juridique.

2. Historique de l'Intelligence Artificielle

L'intelligence artificielle a commencé à émerger dans les années 1950 avec des figures emblématiques comme Alan Turing, qui a proposé le fameux test de Turing pour mesurer la capacité d'une machine à imiter l'intelligence humaine. Les premières décennies ont vu naître des systèmes de résolution de problèmes et des programmes capables de jouer aux échecs. Cependant, ce n'est qu'avec l'avènement des réseaux neuronaux et l'augmentation exponentielle de la puissance de calcul dans les années 2000 que l'IA a connu un essor spectaculaire. Aujourd'hui, les modèles d'IA tels que GPT (Generative Pre-trained Transformer) sont capables de comprendre et de générer du langage naturel avec une grande précision.

3. Exigences et Directives du Projet

Le projet Legal Assistant a été réalisé en suivant les directives suivantes :

3.1 Formation de l'équipe

Le travail a été réalisé en équipe, constituée de quatre membres : **ALAMI Amine**, **ZINE Fahd, AMINE RBII** et **EL BAKKALI Hamza**. Chaque membre a été impliqué de manière équitable dans les différentes tâches de développement.

3.2 Portée du Projet

- **Application alimentée par l'IA** : Le projet a intégré un modèle d'IA pré-entrainé permettant de fournir des réponses précises à des questions juridiques.
- **Connexion à une base de données** : Les documents juridiques sont stockés localement sous forme de fichiers texte et PDF.
- **Problème résolu** : L'application vise à simplifier l'accès aux informations juridiques complexes et à offrir une solution rapide et efficace.

3.3 Technologies Utilisées

1. **Frontend**: HTML5, CSS3, JavaScript, React.js.
2. **Backend** : Flask (Python).
3. **Base de données** : Fichiers locaux structurés.
4. **Outils de test** : Pytest, Postman.

3.4 Sessions de Laboratoire (Détaillées)

Session 1 : Conception

- **Brainstorming** : L'équipe a identifié les problèmes liés à l'accès aux informations juridiques.
- **Design Thinking** : Cette méthode a permis de définir les besoins des utilisateurs.
- **Schémas UML** : Diagramme de cas d'utilisation et diagramme de classes.

Diagramme de cas d'utilisation :

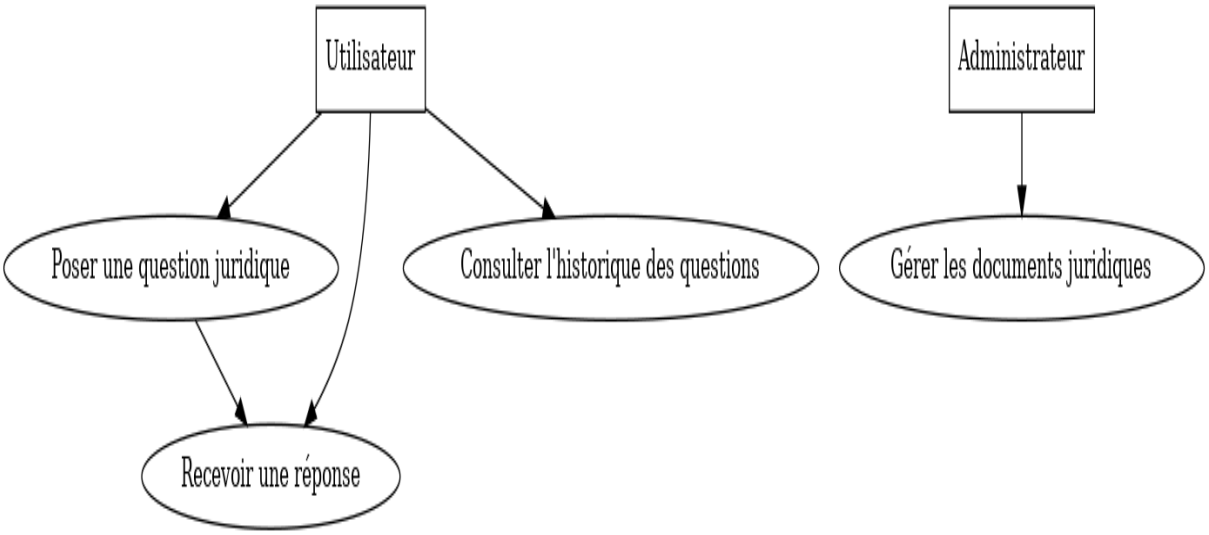
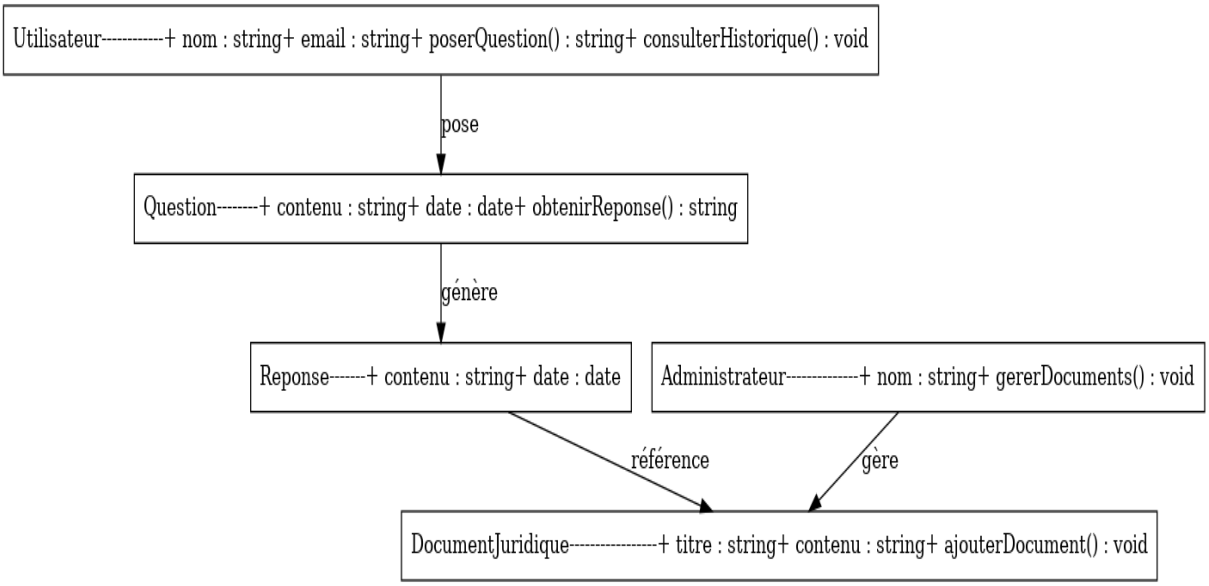


Diagramme de classes :



Session 2 : Développement Backend

- Création des endpoints Flask pour traiter les requêtes de l'application.
- Intégration du modèle d'IA en utilisant des bibliothèques Python spécialisées.
- Gestion des erreurs et des exceptions pour garantir la stabilité du backend

Example endpoint:

```python
@app.route('/query', methods=['POST'])
def query():
    data = request.get_json()
    question = data.get('question')
    response = process_question(question)
    return jsonify({'response': response})
```

Session 3 : Développement Frontend

- Développement de l'interface utilisateur avec React.js.
- Mise en place d'une logique d'interaction fluide entre l'utilisateur et l'application.
- Utilisation de la Fetch API pour communiquer avec le backend.

Exemple de requête Fetch :

```javascript
async function sendMessage() {
  const response = await fetch('/query', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ question: userInput }),
  });
  const data = await response.json();
  displayResponse(data.response);
}
```

Session 4 : Intégration et Tests

- Intégration complète du frontend et du backend.
- Réalisation de tests unitaires avec Pytest.
- Tests fonctionnels simulant des scénarios utilisateur réels.

4. Gestion de Projet

4.1 Outils de Gestion

- **GitHub** : Utilisé pour la gestion du code source et le suivi des versions.
- **Trello** : Employé pour organiser les tâches et suivre l'avancement du projet.
- **Réunions d'équipe** : Des réunions hebdomadaires ont été tenues pour assurer une bonne coordination entre les membres de l'équipe.

5. Analyse des Risques

5.1 Risques Techniques

- **Problèmes de compatibilité** entre le frontend et le backend.
- **Problèmes de performance** lors du traitement des grandes quantités de données juridiques.

5.2 Risques Organisationnels

- **Disponibilité des membres** : Les contraintes de planning ont été gérées grâce à une bonne répartition des tâches et des réunions régulières.

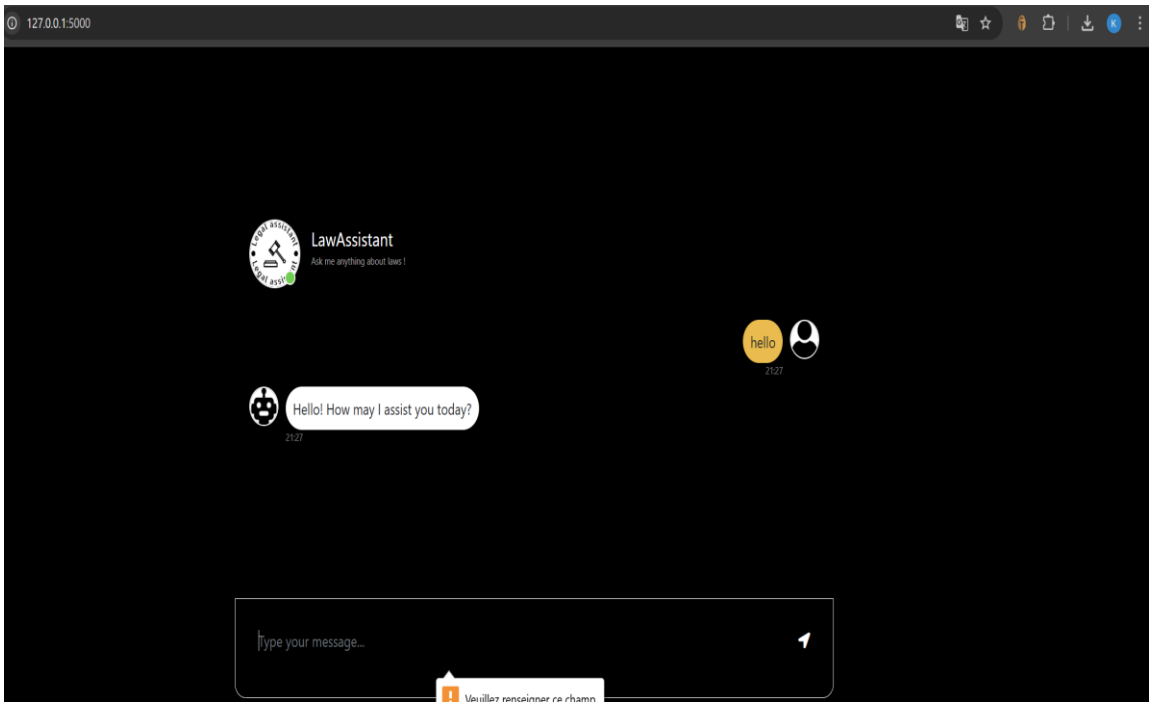
5.3 Solutions Apportées

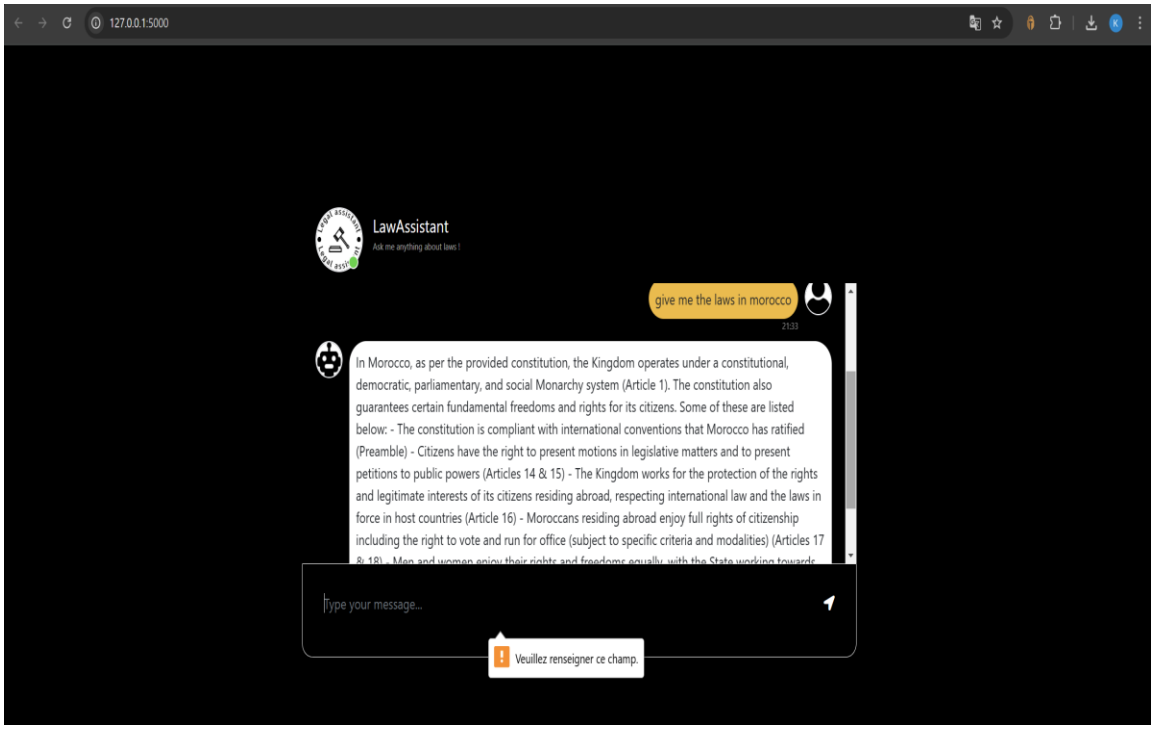
- Compatibilité Flask-CORS pour résoudre les problèmes de compatibilité entre le frontend et le backend.
- Tests réguliers pour détecter et corriger les erreurs rapidement.

6. Plan de Déploiement

6.1 Déploiement Local

- Installation des dépendances avec pip install -r requirements.txt.
- Lancement du serveur Flask avec python app.py.
- Accès à l'application via http://localhost:5000.





6.2 Déploiement sur Heroku

1. **Création de l'application** sur Heroku.
2. **Ajout d'un Procfile** pour spécifier la commande de lancement.
3. **Configuration des pipelines CI/CD** avec GitHub.
4. **Déploiement automatique** à chaque mise à jour du dépôt.

7. Analyse Comparative

Solutions Similaires

- **DoNotPay** : Assistant juridique automatisé, mais orienté vers les utilisateurs anglophones.
- **Legal Robot** : Fournit des explications juridiques, mais repose sur un modèle commercial.

Valeur Ajoutée de Legal Assistant

- **Personnalisation locale** : Entraîné sur des documents juridiques marocains.
- **Gratuit et open source** : Accessible à un large public.
- **Facilité d'utilisation** : Interface intuitive et simple.

8. Conclusion

Le projet **Legal Assistant** constitue une avancée prometteuse dans l'utilisation de l'IA pour automatiser les tâches juridiques et améliorer l'accès aux informations juridiques. Grâce à une architecture modulaire et à l'intégration de technologies modernes, cette application offre une solution efficace et accessible.

Avec des possibilités d'amélioration et d'extension, Legal Assistant peut être déployé à grande échelle pour aider divers utilisateurs dans le domaine juridique.

9. Annexe

Annexe A : Extrait de Code Frontend

```html
<!DOCTYPE html>
<html>
<head>
  <title>Legal Assistant</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header>
    <h1>Legal Assistant Chatbot</h1>
  </header>
  <div class="chatbox">
    <div id="chat-output"></div>
    <input id="userInput" type="text" placeholder="Posez votre question ici...">
    <button onclick="sendMessage()">Envoyer</button>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

Annexe B : Extrait de Code Backend

```python
from flask import Flask, request, jsonify
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

@app.route('/query', methods=['POST'])
def query():
    data = request.get_json()
    question = data.get('question')
    response = process_question(question)
    return jsonify({'response': response})

if __name__ == '__main__':
    app.run(debug=True)
```

10. Sources

1. **Documentation officielle Flask : https://flask.palletsprojects.com**
2. **React.js Documentation : https://reactjs.org/docs/getting-started.html**
3. **Tutoriels et exemples d'utilisation de Flask-CORS : https://flask-cors.readthedocs.io/en/latest/**
4. **Documentation Pytest : https://docs.pytest.org**
5. **Guide Postman pour les tests d'API : https://learning.postman.com/docs/**