Equipe :

AZZOUZ Chaimae

AIDOUN Ibtissam

BOUHNINE Safaa

BOUSSAID Meryeme

Ce document contient le compte rendu de sprint 0 et sprint 1 où on a préparé le workflow de travail de ce projet avant d'entamer l'application et le codage.

Workflow complet : Chatbot de recommandation de cours universitaires avec ontologie pédagogique

Vue d'ensemble du projet :

Utilisateur → Chatbot (NLP) → Agent IA → Ontologie (Web Sémantique) → Recommandations

Architecture complète :

- Phase 1 : Construction de l'ontologie (Web Sémantique)
- Phase 2 : Agent IA avec raisonnement
- Phase 3 : NLP et compréhension du dialogue
- Phase 4 : Intégration du chatbot
- Phase 5 : Interface utilisateur

PHASE 1 : Construction de l'ontologie pédagogique

Étape 1.1 : Modélisation de l'ontologie

Classes principales à créer :

- Cours (Course)
- Étudiant (Student)
- Compétence (Skill)
- Domaine (Domain)
- Prérequis (Prerequisite)
- Niveau (Level: Débutant, Intermédiaire, Avancé)

Relations (Object Properties) :

- aPrerequis (Course → Course)
- enseigneCompetence (Course → Skill)
- appartientADomaine (Course → Domain)
- aNiveau (Course → Level)
- possèdeCompetence (Student → Skill)
- aInteretPour (Student → Domain)
- aSuivi (Student → Course)

Data Properties :

- nomCours (string)
- codeCours (string)

- credits (int)
- duree (int)
- difficulte (int 1-5)
- description (string)

Étape 1.2 : Création de l'ontologie avec Protégé

Outil : Protégé Desktop (gratuit)

1. Télécharger Protégé : https://protege.stanford.edu/
2. Créer une nouvelle ontologie OWL
3. Définir les classes hiérarchiquement
4. Ajouter les propriétés d'objet et de données
5. Créer des règles SWRL (optionnel) pour inférences automatiques

Étape 1.3 : Peuplement de l'ontologie avec des données

- Créer un fichier Python pour peupler l'ontologie
- Ou utiliser un fichier CSV pour automatiser

NOUVEAU : Étape 1.4 : Intégration RDF

Pourquoi RDF ?

- Format standard W3C pour représenter les connaissances
- Permet l'interopérabilité avec d'autres systèmes
- Structure en triplets (Sujet-Prédicat-Objet) très flexible
- Facilite le raisonnement automatique

Conversion de l'ontologie en RDF :

- Exporter l'ontologie Protégé en format RDF/XML ou Turtle (.ttl)
- Définir les vocabulaires RDF personnalisés (COURSE, STUDENT, SKILL)
- Créer un fichier vocabulary.py avec les namespaces RDF

Exemple de représentation RDF d'un cours :

```
course:IA-401 rdf:type course:Course ;
   rdfs:label "Intelligence Artificielle Avancée"@fr ;
   course:codeCours "IA-401" ;
   course:credits "6"^^xsd:integer ;
   course:aPrerequis course:IA-301 .
```

NOUVEAU : Étape 1.5 : Base de Connaissance avec Triple Store

Qu'est-ce qu'une Base de Connaissance ?

- Stockage persistant et évolutif des données RDF
- Requêtes SPARQL optimisées sur des millions de triplets
- Support du raisonnement inférentiel
- Versioning et historique des modifications

Choix du Triple Store : Apache Jena Fuseki (recommandé)

- Open source et gratuit

- Excellente documentation
- Facile à installer avec Docker
- API REST native
- Support complet de SPARQL 1.1

Installation avec Docker :

docker run -p 3030:3030 -e ADMIN_PASSWORD=admin stain/jena-fuseki

Fichiers à créer :

- src/knowledge_base.py : Gestionnaire de connexion au triple store
- scripts/populate_knowledge_base.py : Script de population
- config/fuseki_config.ttl : Configuration Fuseki
- docker/docker-compose.yml : Configuration Docker

PHASE 2 : Agent IA avec raisonnement

Étape 2.1 : Moteur de raisonnement SPARQL

Créer un module de requêtes SPARQL.

Étape 2.2 : Logique d'agent décisionnel

Implémenter les algorithmes de recommandation et de raisonnement.

NOUVEAU : Étape 2.3 : Intégration avec la Base de Connaissance

Modifications du sparql_reasoner.py :

- Remplacer les requêtes sur fichier .ttl par des requêtes vers Fuseki
- Utiliser la classe KnowledgeBase pour toutes les interactions
- Implémenter le cache des requêtes fréquentes

Requêtes SPARQL avancées à implémenter :

- Recherche de cours par domaine
- Chaîne complète des prérequis (récursif)
- Recommandations basées sur compétences et intérêts
- Parcours d'apprentissage optimal
- Vérification automatique des prérequis

PHASE 3 : NLP et compréhension du dialogue

Étape 3.1 : Extraction d'intentions et entités

Utiliser spaCy + modèles pré-entraînés

Étape 3.2 : Gestion du contexte conversationnel

Maintenir l'historique de conversation et le contexte.

NOUVEAU : Étape 3.3 : Extraction d'entités RDF

Mapper les intentions vers des requêtes SPARQL :

- Intention 'recherche_cours' → Requête SELECT sur course:Course
- Intention 'verifier_prerequis' → Requête sur course:aPrerequis
- Intention 'parcours_apprentissage' → Calcul de chemin dans le graphe

Extraire les URIs RDF depuis le texte utilisateur :

- Détecter les codes de cours (ex: IA-401 → course:IA-401)
- Identifier les domaines (ex: 'intelligence artificielle' → course:IntelligenceArtificielle)
- Reconnaître les compétences mentionnées

PHASE 4 : Intégration du Chatbot

Étape 4.1 : Orchestration complète

Connecter tous les modules ensemble.

PHASE 5 : Interface utilisateur

Étape 5.1 : Interface avec Streamlit

Créer une interface simple et rapide avec Streamlit.

NOUVEAU : Étape 5.2 : Visualisation du Graphe RDF

Fonctionnalités de visualisation à ajouter :

- Graphe interactif des prérequis (avec PyVis)
- Dashboard d'exploration de la base de connaissance
- Interface d'administration pour ajouter/modifier des cours
- Visualisation du parcours d'apprentissage recommandé

PHASE 6 : Structure du projet et dépendances

Structure des fichiers :

```
university-chatbot/
│
├── data/
│   ├── cours.csv
│   ├── competences.csv
│   └── university_ontology.ttl
│   └── rdf/                    # □ NOUVEAU
│       ├── courses.rdf
│       ├── skills.rdf
│       └── students.rdf
│
├── src/
│   ├── __init__.py
│   ├── knowledge_base.py       # □ NOUVEAU
```

```
│   ├── sparql_reasoner.py
│   ├── recommendation_agent.py
│   ├── nlp_processor.py
│   ├── conversation_manager.py
│   └── chatbot.py
│
├── scripts/
│   ├── populate_knowledge_base.py      # □ NOUVEAU
│   ├── export_rdf.py                   # □ NOUVEAU
│   └── backup_knowledge_base.py        # □ NOUVEAU
│
├── config/                             # □ NOUVEAU
│   └── fuseki_config.ttl
│
├── docker/                             # □ NOUVEAU
│   ├── docker-compose.yml
│   └── Dockerfile
│
├── app.py
├── requirements.txt
└── README.md
```

Requirements.txt (mis à jour) :

```
# Web Sémantique & RDF
rdflib==7.0.0
owlready2==0.45
SPARQLWrapper==2.0.0
pyshacl==0.25.0

# NLP
spacy==3.7.2
transformers==4.36.0
sentence-transformers==2.2.2

# Chatbot & Interface
streamlit==1.29.0
langchain==0.1.0

# Visualisation
networkx==3.2.1
pyvis==0.3.2
pandas==2.1.4
```

PHASE 7 : Exécution et tests

Étape 7.1 : Setup initial

1. Installer les dépendances
pip install -r requirements.txt
python -m spacy download fr_core_news_md

2. Démarrer Fuseki avec Docker NOUVEAU
docker-compose up -d fuseki

3. Peupler la base de connaissance
python scripts/populate_knowledge_base.py

4. Lancer l'application
streamlit run app.py

Étape 7.2 : Scénarios de test

Test 1 : Recommandation simple
User: "Je cherche des cours en intelligence artificielle"
→ Le chatbot extrait l'intention et le domaine
→ L'agent interroge l'ontologie via SPARQL
→ Retourne les cours d'IA avec prérequis

Test 2 : Vérification de prérequis
User: "Est-ce que je peux suivre le cours IA-401 ?"
→ NLP extrait le code cours
→ Agent vérifie dans l'ontologie si l'étudiant a les prérequis
→ Répond oui/non avec explications

Test 3 : Parcours d'apprentissage
User: "Je veux devenir expert en Machine Learning, par où commencer ?"
→ Agent calcule le chemin dans le graphe de prérequis
→ Retourne une séquence de cours ordonnée

PHASE 8 : Améliorations et extensions

Extensions possibles :

1. Améliorer le NLP

- Ajouter un modèle de similarité sémantique
- Utiliser des embeddings pour trouver des cours similaires
- Gérer les synonymes et variations de langage

2. Enrichir l'ontologie

- Ajouter des données sur les professeurs
- Intégrer les horaires et disponibilités

- Ajouter des avis étudiants

3. Raisonnement avancé

- Implémenter des règles SWRL plus complexes
- Ajouter un moteur d'inférence (Pellet, HermiT)
- Optimisation de parcours avec contraintes

4. Interface

- Visualisation du graphe de cours NOUVEAU
- Dashboard de progression
- Intégration calendrier

Avantages de l'approche RDF et Base de Connaissance

- **Scalabilité :** Le triple store peut gérer des millions de triplets
- **Performance :** Requêtes SPARQL optimisées
- **Flexibilité :** Facile d'ajouter de nouvelles relations
- **Interopérabilité :** Standard W3C compatible avec d'autres systèmes
- **Raisonnement :** Inférence automatique avec règles SWRL
- **Versioning :** Historique des modifications dans la KB
- **API REST :** Accès facile depuis n'importe quelle application

Division des tâches mise à jour (4 membres)

Membre 1 : Spécialiste Ontologie & RDF

Responsabilités :

- Créer l'ontologie complète avec Protégé
- Convertir l'ontologie en format RDF/Turtle
- Configurer le triple store (Fuseki)
- Développer knowledge_base.py
- Créer populate_knowledge_base.py

Membre 2 : Spécialiste NLP

Responsabilités :

- Implémenter nlp_processor.py avec spaCy
- Adapter le NLP pour extraire des entités RDF
- Mapper les intentions vers des requêtes SPARQL
- Développer conversation_manager.py

Membre 3 : Spécialiste Agent IA & Raisonnement

Responsabilités :

- Modifier sparql_reasoner.py pour utiliser knowledge_base.py
- Implémenter le raisonnement avec inférence RDF
- Développer recommendation_agent.py
- Optimiser les requêtes SPARQL complexes

Membre 4 : Spécialiste Interface & Visualisation

Responsabilités :

- Développer l'interface Streamlit (app.py)
- Créer une visualisation du graphe RDF
- Dashboard pour explorer la base de connaissance
- Intégration et tests end-to-end

Conclusion

Ce workflow complet intègre maintenant les technologies RDF et Base de Connaissance pour créer un chatbot de recommandation de cours universitaires puissant, scalable et professionnel.

Les nouvelles fonctionnalités RDF apportent une flexibilité exceptionnelle, des performances optimisées et une conformité aux standards du Web Sémantique.