

# RAPPORT PROJET FIN DE SEMESTRE – DIC1

Fonctionnalités :

- Backend
  - Configuration de la base de données
  - Développement des APIs pour les utilisateurs, sujet et copies
  - Configuration et Intégration de DeepSeek
  - Implémenter la détection de plagiat
  - Authentification
  - Optimisation de l'IA pour Statistiques et rapports
  - Génération de rapports et d'alertes

- 1- Créer le modèle de BD
- 2- Développer les points de sortie des APIs pour l'interface
- 3- Tests des fonctionnalités

- Frontend
  - Conception et développement de l'interface
  - Intégration des APIs backend pour afficher les données
  - Gestion des notifications
  - Création des dashboards
  - Intégration du chatbot

- 1- Imaginer les différentes pages
- 2- Développer les composants React, Angular ou Vue
- 3- Tester les APIs

# INTEGRATION DE DEEPSEEK

L'intégration de DeepSeek nécessite plusieurs étapes :

- Installation d'Ollama

Nous nous rendons sur le site [d'ollama](https://ollama.com) et nous installons la version appropriée (pour Windows/linux/mac).



Discord GitHub Models

Search models

## Download Ollama



macOS



Linux



Windows

Download for Windows

Requires Windows 10 or later

While Ollama downloads, sign up to get notified of new updates.

Une fois l'installation effectuée, nous allons dans notre terminal et exécuter cette commande : ***ollama run deepseek-r1:7b***. Cela nous permettra d'avoir DeepSeek en version locale.

```
Microsoft Windows [version 10.0.26100.3194]
(c) Microsoft Corporation. Tous droits réservés.
```

```
C:\Users\menis>ollama --version
ollama version is 0.5.11
```

```
C:\Users\menis>ollama run deepseek-r1:7b
pulling manifest
pulling 96c415656d37... 17% ██████████
```

```
| 785 MB/4.7 GB 2.6 MB/s 24m42s|
```

## IMPLEMENTATION – CORRECTION IA

Après avoir installé Deepseek-R1 via Ollama, nous pouvons passer à l'étape d'implémentation de la fonctionnalité de correction avec l'IA. Nous avons opté pour une option asynchrone, en l'intégrant dans une API. De ce fait, dès que la correction démarre, la plateforme lance une requête à l'API qui exécute la correction et insère les résultats dans la base de données et renvoie les résultats.

Cette option asynchrone est adoptée pour éviter la complexité et la lourdeur de la plateforme et en même temps permettre à l'utilisateur, en l'occurrence le professeur de naviguer dans le site et d'avoir quand même le résultat de la correction après être revenu.

Pour cela nous avons implémenté l'API en utilisant Python accompagné de Celery, Redis Server et FastAPI. Redis Server s'occupe du stockage cache afin de limiter la perte de données, Celery s'occupe de la logique métier et stocke les résultats dans Redis, Fast API sert d'interface permettant de récupérer les requêtes envoyées par le site.

Voici comment nous avons procédé :

Tout d'abord nous avons le fichier config.py qui contient la requête que l'on enverra à DeepSeek pour lancer la correction.

```
config.py > ...
1  # Configuration des paramètres de DeepSeek
2  OLLAMA_MODEL = "deepseek-r1:7b"
3  PROMPT_TEMPLATE = """
4  Tu es un correcteur avancé. Corrige ce texte en français et attribue-lui une note sur 20.
5  Format de réponse :
6  ---
7  Correction : [Texte corrigé]
8  Note : [Note/20]
9  ---
10 Je répète, tu donnes juste la note, aucun commentaire.
11 Texte :
12 {texte}
13 """
14 |
```

Ensuite, il est nécessaire d'implémenter le fichier qui permettra d'upload les PDFs des examens :

```

pdf_loader.py > ...
1  import fitz
2  import os
3
4  def extract_text_from_pdf(pdf_path):
5      """Extrait le texte d'un fichier PDF."""
6      text = ""
7      with fitz.open(pdf_path) as doc:
8          for page in doc:
9              text += page.get_text("text") + "\n"
10     return text
11
12     Tabnine | Edit | Test | Explain | Document
13     def load_pdfs_from_folder(folder_path):
14         """Charge tous les PDFs d'un dossier et extrait leur texte."""
15         pdf_texts = {}
16         for file_name in os.listdir(folder_path):
17             if file_name.endswith(".pdf"):
18                 file_path = os.path.join(folder_path, file_name)
19                 pdf_texts[file_name] = extract_text_from_pdf(file_path)
20         return pdf_texts

```

Ensuite, nous configurons le client deepseek qui permettra de récupérer la réponse de DeepSeek et d'y extraire la note importée pour l'avoir dans la base de données.

```

deepseek_client.py > ...
1  import ollama
2  import re
3  from config import PROMPT_TEMPLATE
4  import difflib
5  import os
6  from database import get_copie_id, insert_plagiats
7
8  Tabnine | Edit | Test | Explain | Document
9  def correct_text_with_deepseek(text):
10     """Envoie un texte à DeepSeek pour correction et notation, en nettoyant la réponse."""
11     prompt = PROMPT_TEMPLATE.format(text=text)
12     response = ollama.chat(model="deepseek-r1:7b", messages=[{"role": "user", "content": prompt}])
13     corrected_content = response["message"]["content"]
14
15     # Suppression du contenu entre <think> et </think>
16     cleaned_content = re.sub(r"<think>.*?</think>", "", corrected_content, flags=re.DOTALL)
17
18     return cleaned_content
19

```

```

40 def compare_texts_with_deepseek(text1, text2):
41     """Compare deux textes et détecte le plagiat."""
42     prompt = PLAGIARISM_PROMPT_TEMPLATE.format(text1=text1, text2=text2)
43
44     response = ollama.chat(model="deepseek-r1:7b", messages=[{"role": "user", "content": prompt}])
45     content = response["message"]["content"]
46
47     cleaned_content = re.sub(r"<think>.*?</think>", "", content, flags=re.DOTALL)
48
49     match = re.search(r"Taux de similarité : (\d+)%", cleaned_content)
50     taux_similarite = int(match.group(1)) if match else 0
51
52     return cleaned_content, taux_similarite
53

```

Par la même occasion, nous avons implémenté le détecteur de plagiat pour comparer chaque fichier 2 à 2 afin de déterminer s'il existe des similitudes.

```

Tabnine | Edit | Test | Explain | Document
def detect_plagiarism(texts):
    """
    Compare chaque fichier 2 à 2 et détecte le plagiat.
    Insère les cas de plagiat dans la base de données en une seule requête.
    Retourne un dictionnaire contenant les taux de similarité et les rapports.
    """
    plagiarism_results = []
    plagiat_data = [] # Liste pour stocker les cas de plagiat avant insertion

    file_names = list(texts.keys())

    for i in range(len(file_names)):
        for j in range(i + 1, len(file_names)):
            file1, file2 = file_names[i], file_names[j]
            text1, text2 = texts[file1], texts[file2]

            # Calcul du taux de similarité
            similarity_ratio = difflib.SequenceMatcher(None, text1, text2).ratio() * 100

            if similarity_ratio > 30:
                report = f"Plagiat détecté entre {file1} et {file2}.\nTaux de similarité : {similarity_ratio}%"

                # Sauvegarder le rapport dans un fichier
                os.makedirs("plagiarism_reports", exist_ok=True)
                report_path = os.path.join("plagiarism_reports", f"{file1}_vs_{file2}.txt")
                with open(report_path, "w", encoding="utf-8") as f:
                    f.write(report)

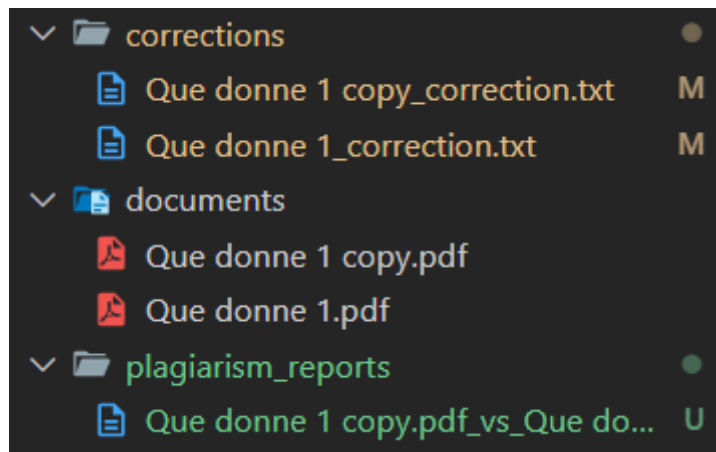

```

```

1
2     # Ajout des résultats dans la liste
3     plagiarism_results.append({
4         "file1": file1,
5         "file2": file2,
6         "taux_similarite": similarity_ratio,
7         "rapport": report
8     })
9
10    # Récupérer l'ID de la copie
11    copie_id = get_copie_id(file1)
12    if copie_id:
13        plagiat_data.append((copie_id, similarity_ratio, report, int(similarity_ratio > 70)))
14
15    # Insérer tous les cas de plagiat en une seule requête
16    if plagiat_data:
17        insert_plagiats(plagiat_data)
18
19    return plagiarism_results

```

Par la suite, il faudra sauvegarder les résultats dans des fichiers. Nous les avons mis pour le moment dans des fichiers textes. Les résultats de correction seront dans le répertoire corrections/ et les résultats de plagiats seront dans le répertoire plagiarism\_reports/



Le code correspondant est présenté ci-dessous :

```
utils.py > extract_note_from_file
1  import os
2  import re
3
4  Tabnine | Edit | Test | Explain | Document
5  def save_correction(file_name, corrected_text, output_folder="corrections"):
6      """Sauvegarde la correction d'un fichier PDF sous format texte."""
7      os.makedirs(output_folder, exist_ok=True)
8
9      output_path = os.path.join(output_folder, file_name.replace(".pdf", "_correction.txt"))
10
11     with open(output_path, "w", encoding="utf-8") as f:
12         f.write(corrected_text)
13
14     print(f"Correction sauvegardée : {output_path}")
15
16     Tabnine | Edit | Test | Explain | Document
17     def extract_note_from_file(file_path):
18         """Extrait la note d'un fichier de correction."""
19         with open(file_path, "r", encoding="utf-8") as f:
20             content = f.read()
21
22             match = re.search(r"Note\s*:\s*(\d+)(?:/20)?", content)
23
24             if match:
25                 return int(match.group(1))
26             return None
```

Nous avons par la suite implémenté un fichier database.py qui contient les méthodes nécessaires pour insérer les résultats dans les tables correspondantes de la base de données.

```

database.py > insert_plagiats
1  import mysql.connector
2  import os
3  import json
4
5  # Connexion à la base de données
  Tabnine | Edit | Test | Explain | Document
6  > def get_db_connection(): ...
16
  Tabnine | Edit | Test | Explain | Document
17 > def get_copie_id(file_name): ...
25
  Tabnine | Edit | Test | Explain | Document
26 > def insert_correction(copie_id, note): ...
47
  Tabnine | Edit | Test | Explain | Document
48 > def update_distribution(examen_id, note): ...
75
  Tabnine | Edit | Test | Explain | Document
76 > def update_statistiques(examen_id, cur): ...
101
  Tabnine | Edit | Test | Explain | Document
102 > def get_examen_id_from_copie(copie_id): ...
115
  Tabnine | Edit | Test | Explain | Document
116 > def insert_plagiats(plagiat_list): ...
135
136

```

Maintenant que tout ceci a été implémenté, il est temps de définir la logique de Celery et de FastAPI permettant de gérer tout cela de manière asynchrone.

Le fichier Celery gère la logique métier et stocke ses résultats dans la mémoire Redis.

```

celery_worker.py > ...
1  from celery import Celery
2  from deepseek_client import compare_texts_with_deepseek, correct_text_with_deepseek,
   detect_plagiarism
3  from pdf_loader import load_pdfs_from_folder
4  from utils import save_correction, extract_note_from_file
5  from database import get_copie_id, get_db_connection, insert_correction, update_distribution,
   update_statistiques, get_examen_id_from_copie
6  import os
7
8  # Configuration de Celery
9  celery_app = Celery(
10     "tasks",
11     broker="redis://localhost:6380/0",
12     backend="redis://localhost:6380/0",
13
14 )
15
16 celery_app.conf.result_backend = "redis://localhost:6380/0"
17
18 celery_app.conf.update (
19     result_backend="redis://localhost:6380/0",
20     task_serializer="json",
21     accept_content=["json"],
22     result_expires=3600,
23 )
24
25

```

Après cette configuration, nous passons à l'implémentation de la méthode. La première partie concerne uniquement la correction :



```

celery_worker.py > ...
27 def process_pdfs():
28     """Tâche de correction et détection de plagiat exécutée en arrière-plan."""
29     pdf_texts = load_pdfs_from_folder("documents")
30     notes = {}
31
32     for file_name, text in pdf_texts.items():
33         print(f"Correction du fichier : {file_name}...")
34
35         corrected_text = correct_text_with_deepseek(text)
36         save_correction(file_name, corrected_text)
37
38         correction_file_path = os.path.join("corrections", file_name.replace(".pdf", "_correction.
txt"))
39         note = extract_note_from_file(correction_file_path)
40
41         if note is not None:
42             notes[file_name] = note
43             print(f>Note extraite pour {file_name} : {note}/20")
44
45             copie_id = get_copie_id(file_name)
46             if copie_id:
47                 conn = get_db_connection()
48                 cur = conn.cursor()
49
50                 insert_correction(copie_id, note)
51                 examen_id = get_examen_id_from_copie(copie_id)
52
53                 if examen_id:
54                     update_distribution(examen_id, note)
55                     update_statistiques(examen_id, cur)
56
57                 cur.close()
58                 conn.close()

```

La seconde partie permet de gérer la logique de plagiat :

```

59
60 plagiarism_results = detect_plagiarism(pdf_texts)
61 for result in plagiarism_results:
62     file1, file2 = result["file1"], result["file2"]
63     similarity = result["taux_similarite"]
64     report = result["rapport"]
65
66     copie_id = get_copie_id(file1)
67
68     if copie_id:
69         conn = get_db_connection()
70         cur = conn.cursor()
71
72         cur.execute("""
73             INSERT INTO plagiat (id_copie, taux_similarite, rapport, alerte)
74             VALUES (%s, %s, %s, %s)
75             """, (copie_id, similarity, report, 1 if similarity > 50 else 0))
76
77         conn.commit()
78         cur.close()
79         conn.close()
80
81         print(f"🔍 Plagiat detecte entre {file1} et {file2}, taux: {similarity:.2f}%")
82
83     return {"notes": notes, "plagiat": plagiarism_results}
84

```

Le dernier fichier concerne la logique de FastAPI. Là dedans, nous gérons toute la logique avec leurs points de sorties. Nous gérons d'abord les importations et les configurations :

```
api.py > ...
1  from fastapi import FastAPI, HTTPException
2  from celery.result import AsyncResult
3  from celery_worker import detect_plagiarism, process_pdfs
4  from fastapi.middleware.cors import CORSMiddleware
5  from celery_worker import celery_app
6  from database import get_db_connection
7  import redis
8
9  redis_client = redis.StrictRedis(host='localhost', port=6380, db=0)
10
11 app = FastAPI()
12
13 app.add_middleware(
14     CORSMiddleware,
15     allow_origins=["*"],
16     allow_credentials=True,
17     allow_methods=["*"],
18     allow_headers=["*"],
19 )
```

Chaque point de sortie avec son travail comme montré ci-dessous :

```
21 @app.post("/start_correction")
22 def start_correction():
23     """Lance la correction en arrière-plan."""
24     task = process_pdfs.apply_async() # Démarre la tâche Celery
25     print(f"Tâche lancée : {task.id}") # Log de la tâche
26     return {"message": "Correction en cours", "task_id": task.id}
27
28 Tabnine | Edit | Test | Explain | Document
29 @app.get("/task_status/{task_id}")
30 def task_status(task_id: str):
31     """Récupère l'état d'une tâche Celery."""
32     task = celery_app.AsyncResult(task_id)
33     print(f"Statut de la tâche {task_id}: {task.state}") # Log du statut de la tâche
34
35     if task.state == "PENDING":
36         return {"task_id": task_id, "status": "En attente"}
37     elif task.state == "STARTED":
38         return {"task_id": task_id, "status": "En cours de traitement"}
39     elif task.state == "SUCCESS":
40         redis_client.rpush("correction_results", str({"task_id": task_id, "result": task.result}))
41         return {"task_id": task_id, "status": "Terminé", "result": task.result}
42     elif task.state == "FAILURE":
43         return {"task_id": task_id, "status": "Échec", "result": task.result}
44     else:
45         return {"task_id": task_id, "status": task.state}
```

```

Tabnine | Edit | Test | Explain | Document
46 @app.get("/get_corrections")
47 def get_corrections():
48     """Récupère tous les résultats de correction stockés dans Redis."""
49     results = redis_client.lrange("correction_results", 0, -1) # Récupère tous les résu
50     results = [eval(res.decode('utf-8')) for res in results] # Convertir les résultats
51     return {"corrections": results}
52
Tabnine | Edit | Test | Explain | Document
53 @app.post("/start_plagiarism_detection")
54 def start_plagiarism_detection():
55     """Lance la détection de plagiat en arrière-plan."""
56     task = detect_plagiarism.apply_async()
57     return {"message": "Détection de plagiat en cours", "task_id": task.id}
58
Tabnine | Edit | Test | Explain | Document
59 @app.get("/result/{task_id}")
60 def get_result(task_id: str):
61     """Récupère le résultat d'une tâche Celery."""
62     task = AsyncResult(task_id)
63
64     if task.state == "SUCCESS":
65         return {"task_id": task_id, "status": "Terminé", "result": task.result}
66
67     return {"task_id": task_id, "status": task.state}
68

```

```

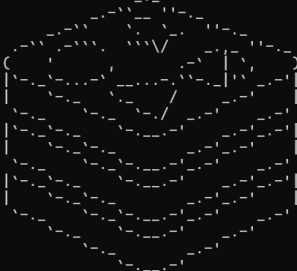
Tabnine | Edit | Test | Explain | Document
69 @app.get("/get_submissions")
70 def get_submissions():
71     """Récupère la liste des fichiers soumis depuis la base de données."""
72     conn = get_db_connection()
73     cur = conn.cursor()
74
75     try:
76         cur.execute("SELECT fichier_pdf FROM copies")
77         files = [row[0] for row in cur.fetchall()]
78         return {"files": files}
79     except Exception as e:
80         raise HTTPException(status_code=500, detail=str(e))
81     finally:
82         cur.close()
83         conn.close()

```

Ceci nous permet d'avoir un résultat concret sur la correction de copies. A noter que nous avons juste fait en sorte que DeepSeek corrige les copies par elle-même en suivant sa propre logique, sans upload un fichier corrigé dont il se basera pour effectuer les comparaisons de correction. Nous aurons finalement comme résultat suivant :

Nous avons lancé redis-server et Celery :

```
PS C:\Users\menis\OneDrive\Documents\MES_FICHIERS_DIC\MES_FICHIERS_DIC1\SEMESTRE 1\SGBD\PROJET\Project\deepseek_correction> redis-server C:\Users\menis\redis.conf
```



```
Redis 3.0.504 (00000000/0) 64 bit

Running in standalone mode
Port: 6380
PID: 23276

http://redis.io
```

```
[23276] 16 Mar 13:56:39.812 # Server started, Redis version 3.0.504
[23276] 16 Mar 13:56:39.813 * The server is now ready to accept connections on port 6380
```

```
PS C:\Users\menis\OneDrive\Documents\MES_FICHIERS_DIC\MES_FICHIERS_DIC1\SEMESTRE 1\SGBD\PROJET\Project\deepseek_correction> celery -A celery_worker.celery_app worker --pool=solo --loglevel=info
```

```
----- celery@MADENIYOU v5.4.0 (opalescent)
--- ***** -----
-- ***** --- Windows-10-10.0.26100-SP0 2025-03-16 13:57:54
- *** --- * ---
- ** ----- [config]
- ** ----- .> app: tasks:0x1fc2827fffd0
- ** ----- .> transport: redis://localhost:6380/0
- ** ----- .> results: redis://localhost:6380/0
- *** --- * --- .> concurrency: 12 (solo)
-- ***** --- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
----- [queues]
. > celery exchange=celery(direct) key=celery

[tasks]
. celery_worker.process_pdfs

[2025-03-16 13:57:54,701: WARNING/MainProcess] C:\Python311\Lib\site-packages\celery\worker\consumer\consumer.py:508: CPendingDeprecationWarning: The broker_connection_retry configuration setting will no longer determine whether broker connection retries are made during startup in Celery 6.0 and above. If you wish to retain the existing behavior for retrying connections on startup, you should set broker_connection_retry_on_startup to True.
warnings.warn(

[2025-03-16 13:57:54,715: INFO/MainProcess] Connected to redis://localhost:6380/0
[2025-03-16 13:57:54,716: WARNING/MainProcess] C:\Python311\Lib\site-packages\celery\worker\consumer\consumer.py:508: CPendingDeprecationWarning: The broker_connection_retry configuration setting will no longer determine whether broker connection retries are made during startup in Celery 6.0 and above. If you wish to retain the existing behavior for retrying connections on startup, you should set broker_connection_retry_on_startup to True.
warnings.warn(

[2025-03-16 13:57:54,719: INFO/MainProcess] mingle: searching for neighbors
[2025-03-16 13:57:55,755: INFO/MainProcess] mingle: all alone
[2025-03-16 13:57:55,779: INFO/MainProcess] celery@MADENIYOU ready.
```

De ce fait, nous pouvons maintenant lancer FastAPI via uvicorn pour recevoir les éventuelles requêtes :

```
PS C:\Users\menis\OneDrive\Documents\MES_FICHIERS_DIC\MES_FICHIERS_DIC1\SEMESTRE 1\SGBD\PROJET\Project\deepseek_correction> uvicorn api:app --reload
```

```
INFO: Will watch for changes in these directories: ['C:\\Users\\menis\\OneDrive\\Documents\\MES_FICHIERS_DIC\\MES_FICHIERS_DIC1\\SEMESTRE 1\\SGBD\\PROJET\\Project\\deepseek_correction']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [8952] using StatReload
INFO: Started server process [1428]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Si nous lançons donc une requête, nous aurons ceci :

```
PS C:\Users\menis\OneDrive\Documents\MES_FICHIERS_DIC\MES_FICHIERS_DIC1\SEMESTRE 1\SGBD\PROJET\Project\deepseek_correction> uvicorn api:app --reload
```

```
INFO: Will watch for changes in these directories: ['C:\\Users\\menis\\OneDrive\\Documents\\MES_FICHIERS_DIC\\MES_FICHIERS_DIC1\\SEMESTRE 1\\SGBD\\PROJET\\Project\\deepseek_correction']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [29588] using StatReload
INFO: Started server process [29004]
INFO: Waiting for application startup.
INFO: Application startup complete.
Statut de la tâche 5a3b0fcf-cec4-4227-aca3-8ebe7b8b2615: SUCCESS
INFO: 127.0.0.1:51546 - "GET /task_status/5a3b0fcf-cec4-4227-aca3-8ebe7b8b2615 HTTP/1.1" 200 OK
Tâche lancée : ea79b176-b89d-4b6d-aa48-9c6166692198
INFO: 127.0.0.1:51546 - "POST /start_correction HTTP/1.1" 200 OK
```

```

[tasks]
. celery_worker.process_pdfs

[2025-03-16 13:57:54,701: WARNING/MainProcess] C:\Python311\Lib\site-packages\celery\worker\consumer.py:508: CPendingDeprecationWarning: The broker
_connection_retry configuration setting will no longer determine
whether broker connection retries are made during startup in Celery 6.0 and above.
If you wish to retain the existing behavior for retrying connections on startup,
you should set broker_connection_retry_on_startup to True.
warnings.warn(

[2025-03-16 13:57:54,715: INFO/MainProcess] Connected to redis://localhost:6380/0
[2025-03-16 13:57:54,716: WARNING/MainProcess] C:\Python311\Lib\site-packages\celery\worker\consumer.py:508: CPendingDeprecationWarning: The broker
_connection_retry configuration setting will no longer determine
whether broker connection retries are made during startup in Celery 6.0 and above.
If you wish to retain the existing behavior for retrying connections on startup,
you should set broker_connection_retry_on_startup to True.
warnings.warn(

[2025-03-16 13:57:54,719: INFO/MainProcess] mingle: searching for neighbors
[2025-03-16 13:57:55,755: INFO/MainProcess] mingle: all alone
[2025-03-16 13:57:55,779: INFO/MainProcess] celery@MADENIYOU ready.
[2025-03-16 14:04:12,586: INFO/MainProcess] Task celery_worker.process_pdfs[5a3b0fcf-cec4-4227-aca3-8ebe7b8b2615] received
[2025-03-16 14:04:12,732: WARNING/MainProcess] Correction du fichier : Que donne 1 copy.pdf...
[2025-03-16 14:06:24,750: INFO/MainProcess] HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"
[2025-03-16 14:06:24,761: WARNING/MainProcess] Correction sauvegardée : corrections\Que donne 1 copy_correction.txt
[2025-03-16 14:06:24,787: WARNING/MainProcess] Note extraite pour Que donne 1 copy.pdf : 9/20
[2025-03-16 14:06:24,817: INFO/MainProcess] package: mysql.connector.plugins
[2025-03-16 14:06:24,817: INFO/MainProcess] plugin_name: mysql_native_password
[2025-03-16 14:06:24,824: INFO/MainProcess] AUTHENTICATION_PLUGIN_CLASS: MySQLNativePasswordAuthPlugin
[2025-03-16 14:06:24,978: WARNING/MainProcess] Correction du fichier : Que donne 1.pdf...
[2025-03-16 14:07:48,592: INFO/MainProcess] HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"
[2025-03-16 14:07:48,594: WARNING/MainProcess] Correction sauvegardée : corrections\Que donne 1_correction.txt
[2025-03-16 14:07:48,606: WARNING/MainProcess] Note extraite pour Que donne 1.pdf : 17/20
[2025-03-16 14:07:48,685: WARNING/MainProcess] 🚩 Plagiat detecte entre Que donne 1 copy.pdf et Que donne 1.pdf, taux: 100.00%.
[2025-03-16 14:07:48,700: INFO/MainProcess] Task celery_worker.process_pdfs[5a3b0fcf-cec4-4227-aca3-8ebe7b8b2615] succeeded in 216.1089999999992s: {'notes'
: {'Que donne 1 copy.pdf': 9, 'Que donne 1.pdf': 17}, 'plagiat': [{'file1': 'Que donne 1 copy.pdf', 'file2': 'Que donne 1.pdf', 'taux_similarite': 100.0, 'r
apport': 'Plagiat détecté entre Que donne 1 copy.pdf et Que donne 1.pdf.
Taux de similarité : 100.00%
'}]}

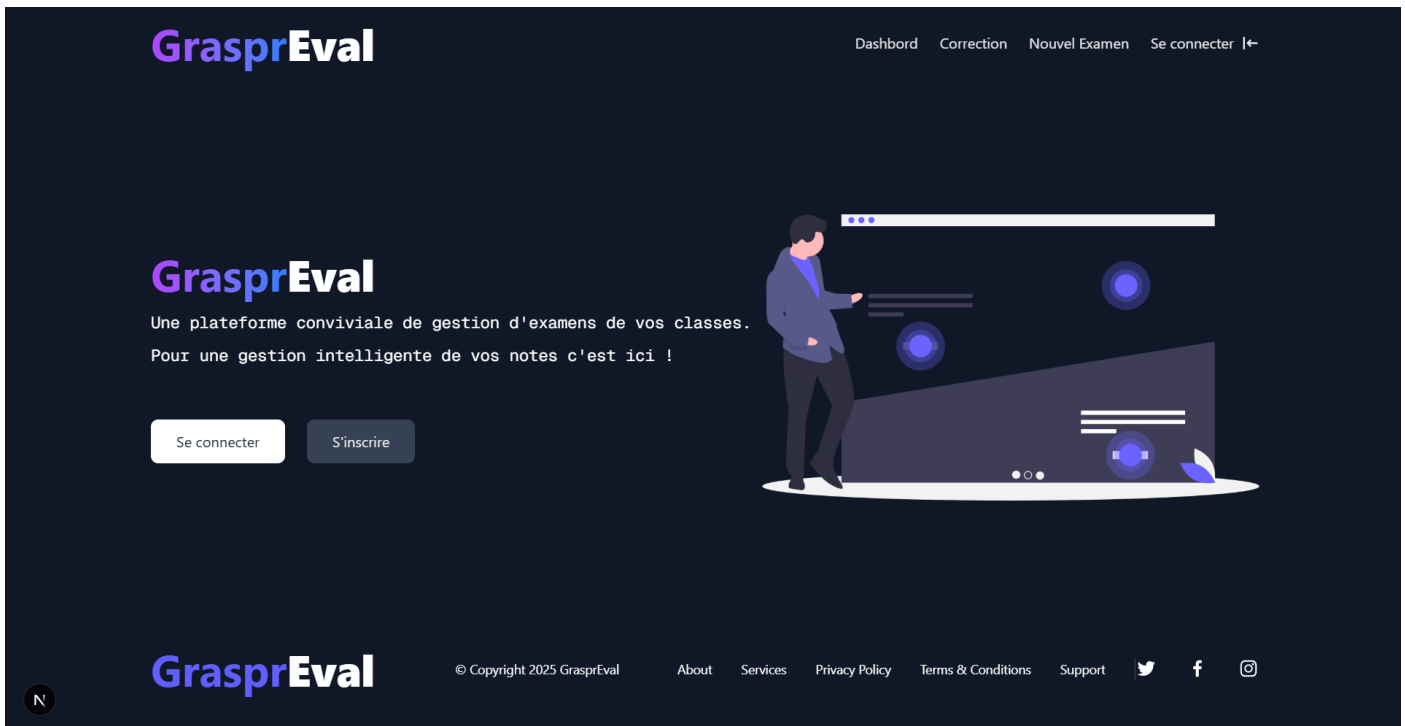
```

Le problème que nous rencontrons souvent est la communication entre les deux parties ainsi que la gestion des performances et le déploiement car DeepSeek-R1 est utilisé en local.

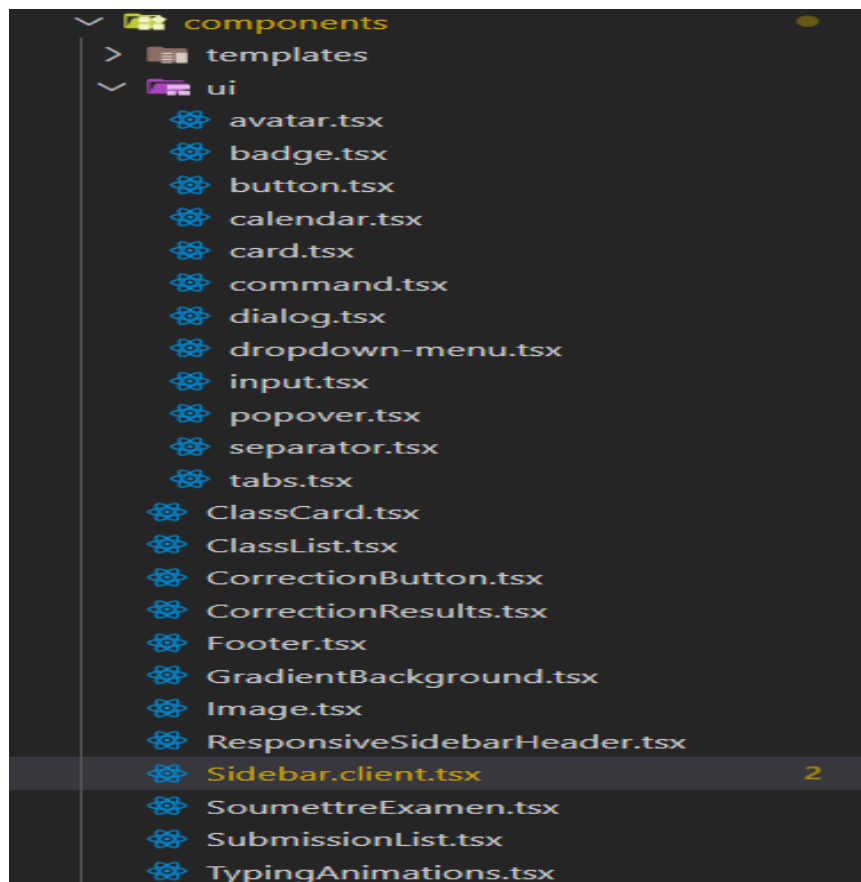
Maintenant que ceci a été fait, nous pouvons commencer à regarder la partie globale, interface de la plateforme : GrasprEval.

## GrasprEval

Notre projet GrasprEval est basé sur NextJS afin de centraliser le frontend et le backend. Nous avons mis en place plusieurs routes et pages, parmi elles la page d'accueil se présentant de la sorte.



Le projet est basé sur plusieurs composants définis dans le répertoire components.



Les pages de connexion et d'inscription n'ont pas encore été gérées. Mais nous avons eu à avoir pour le moment la page où le professeur peut soumettre un nouvel examen ; présentée ci-dessous :

## Formulaire de Soumission

### Nom de l'examen

Entrez le nom de l'examen

### Description

Entrez la description de l'examen

### Type d'examen

Sélectionnez un type



### Date limite

jj/mm/aaaa



### Classes concernées

☐ DUT1

☐ DST1

☐ DUT2

☐ DIC1

☐ DIC2

☐ DIC3

### Ajouter une nouvelle classe (si non présente)

Entrez le nom de la nouvelle classe

Ajouter

### Fichier de l'examen

Choisir un fichier Aucun fichier n'a été sélectionné

Soumettre

## Informations supplémentaires



GrasprEval

Sénégal

Dakar

Université Cheikh Anta Diop

Ecole Supérieure Polytechnique BP 21000, Dakar

Son code correspondant est mis dans le composant SoumettreExamen.tsx dont une partie du code est présentée ci-dessous :



```

6  const SoumettreExamen = () => {
53  return (
54      <div className="bg-gray-900 min-h-screen">
55          /* Section du formulaire avec dégradé */
56          <div className="bg-gray-900">
57              <div className="px-4 mx-auto sm:px-6 lg:px-8 max-w-7xl">
58                  <div className="max-w-2xl mx-auto text-center pt-10 sm:pt-16 lg:pt-24">
59                      <h2 className="text-3xl font-bold leading-tight text-white sm:text-4xl lg:text-5xl">
60                          Soumettre un Examen
61                      </h2>
62                      <p className="max-w-xl mx-auto mt-4 text-base leading-relaxed text-white">
63                          Remplissez le formulaire ci-dessous pour soumettre un nouvel examen.
64                      </p>
65                  </div>
66              </div>
67          </div>
68      </div>
69      /* Contenu principal avec fond noir */
70      <div className="px-4 mx-auto sm:px-6 lg:px-8 max-w-7xl">
71          <div className="max-w-6xl mx-auto mt-12 overflow-hidden bg-white rounded-md shadow-md lg:m
72              <div className="grid items-stretch grid-cols-1 lg:grid-cols-5">
73                  <div className="lg:col-span-3">
74                      <div className="p-6 sm:p-10">
75                          <h3 className="text-2xl font-semibold text-black">Formulaire de Soumission</h3>
76                      </div>
77                      <form onSubmit={handleSubmit} className="mt-8">
78                          <div className="grid grid-cols-1 sm:grid-cols-2 gap-x-5 gap-y-4">
79                              /* Champ pour le nom de l'examen */
80                              <div>
81                                  <label htmlFor="nomExamen" className="text-base font-medium text-gray-900">
82                                      Nom de l'examen
83                                  </label>
84                                  <div className="mt-2.5 relative">

```

Le backend du formulaire permettant d'insérer les informations dans la base de données est gérée par la fonction `handleSubmit` toujours dans le même composant :

```

const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    const formData = new FormData();
    formData.append('nomExamen', nomExamen);
    formData.append('description', description);
    formData.append('typeExamen', typeExamen);
    formData.append('dateLimite', dateLimite);
    formData.append('classes', JSON.stringify(selectedClasses));
    if (fichier) {
        formData.append('fichier', fichier);
    }

    try {
        const result = await soumettreExamen(formData); // Appeler l'action côté serveur
        toast.success(result.message);
    } catch (error) {
        console.error('Erreur :', error);
        alert('Erreur lors de la soumission de l'examen');
    }
};

```



La fonction de soumission d'examen a une partie du code donnée ici :

```
8 export async function soumettreExamen(formData: FormData) {
9   const nomExamen = formData.get('nomExamen') as string;
10  const description = formData.get('description') as string;
11  const typeExamen = formData.get('typeExamen') as string;
12  const dateLimite = formData.get('dateLimite') as string;
13  const classes = formData.get('classes') as string;
14  const fichier = formData.get('fichier') as File;
15
16  try {
17    // Connexion à la base de données
18    const db = await mysql.createConnection({
19      host: 'mysql-n0reyni.alwaysdata.net',
20      user: 'n0reyni_sall',
21      password: 'passer123',
22      database: 'n0reyni_bd',
23    });
24
25    const classesArray = JSON.parse(classes);
26
27    // Gérer l'upload du fichier
28    let fichierPath = null;
29    if (fichier) {
30      const uploadsDir = path.join(process.cwd(), 'public', 'uploads');
31      const fileName = `${Date.now()}-${fichier.name}`; // Nom unique pour éviter les conflits
32      fichierPath = path.join('/uploads', fileName); // Chemin relatif pour la base de données
33
34      // Écrire le fichier dans le dossier `public/uploads`
35      const fileBuffer = await fichier.arrayBuffer();
36      await fs.writeFile(path.join(uploadsDir, fileName), Buffer.from(fileBuffer));
37    }
38  }
```

Après cela nous avons implémenté l'interface du Dashboard où le professeur peut vérifier les statistiques de ses classes. Pour le moment nous avons mis des données lambda pour donner une idée de l'interface :



Les composants ici sont subdivisés en plusieurs composants pour une meilleure gestion des composants en vue d'une éventuelle mise à jour.

En dehors de cela, nous avons eu à implémenter les fonctionnalités de 3 composants. Celles-ci permettent de communiquer avec l'API de correction et de plagiat et récupérer l'état des tâches en fonction de la progression. Ces composants sont CorrectionButton.tsx, CorrectionResults.tsx et SubmissionList.tsx dont une partie de leurs codes sont présentées ci-dessous :

```
src > components > CorrectionButton.tsx > ...
1  import { useState } from "react";
2
3  interface CorrectionButtonProps {
4    setCorrectionResult: (result: any) => void;
5    taskId: string | null;
6    setTaskId: (taskId: string) => void;
7  }
8
9  Tabnine | Edit | Test | Explain | Document
10 export default function CorrectionButton({
11   setCorrectionResult,
12   taskId,
13   setTaskId,
14 }: CorrectionButtonProps) {
15   const [isLoading, setIsLoading] = useState(false);
16   const [message, setMessage] = useState<string | null>(null);
17
18   const startCorrection = async () => {
19     setIsLoading(true);
20     setMessage(null);
21
22     try {
23       const response = await fetch("http://127.0.0.1:8000/start_correction", {
24         method: "POST",
25       });
26
27       if (!response.ok) throw new Error("Erreur lors du démarrage de la correction");
28
29       const data = await response.json();
30       console.log("Tâche démarrée avec ID :", data.task_id);
31       setTaskId(data.task_id);
32       setMessage("Correction démarrée avec succès !");
33     } catch (error) {
```

```
src > components > CorrectionResults.tsx > CorrectionResults
1  Tabnine | Edit | Test | Explain | Document
2  export default function CorrectionResults({ correctionResult }: { correctionResult: any }) {
3    return (
4      <div className="w-1/4 bg-gray-100 p-4">
5        <h2 className="font-bold mb-4">Résultat</h2>
6        <div className="bg-white p-4 shadow rounded min-h-[100px]">
7          {correctionResult ? (
8            <pre>{JSON.stringify(correctionResult, null, 2)}</pre> // Affiche le résultat formaté
9          ) : (
10            "Aucun résultat encore..."
11          )}
12        </div>
13      </div>
14    );
15  }
```

src > components > SubmissionList.tsx > ...

```
1 import { useEffect, useState } from "react";
2
3 Tabnine | Edit | Test | Explain | Document
4 export default function SubmissionList() {
5   const [submissions, setSubmissions] = useState<string[]>([]);
6
7   useEffect(() => {
8     const fetchSubmissions = async () => {
9       try {
10         const response = await fetch("http://127.0.0.1:8000/get_submissions");
11         const data = await response.json();
12         setSubmissions(data.files);
13       } catch (error) {
14         console.error("Erreur lors de la récupération des soumissions :", error);
15       }
16     };
17     fetchSubmissions();
18   }, []);
19
20   return (
21     <div className="sidebar">
22       <h2 className="font-bold mb-4">Copies soumises</h2>
23       <ul>
24         {submissions.map((sub, index) => (
25           <li key={index} className="card">
26             {sub}
27           </li>
28         ))}
29       </ul>
30     </div>
31   );
32 }
```

Nous avons aussi commencé la conception de l'interface pour la correction, pour le moment elle se présente comme suit :

- Accueil
- Dashboard
- Nouvel Examen

Classes

- DUT1
- DUT2
- DST1
- DST2
- DIC1
- DIC2
- DIC3
- L1
- L2
- L3

N



DUT1

Enseignant :



DUT2

Enseignant :



DST1

Enseignant :



DST2

Enseignant :



DIC1

Enseignant :



DIC2

Enseignant :



Copilot

