

Introduction



Dans ce projet, il était demandé de transformer une BDD type SQL (relationnelle) en type No-SQL (non-relationnelle avec MongoDB).



Notre équipe technique



Bernardo



Gael



Fabrice.B



Sommaire

1- Contexte

2- Architecture

3- Migrateur

4- Gestion des notes

5- Technologies

6- Fonctionnalités

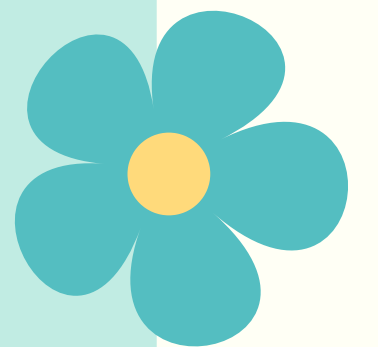
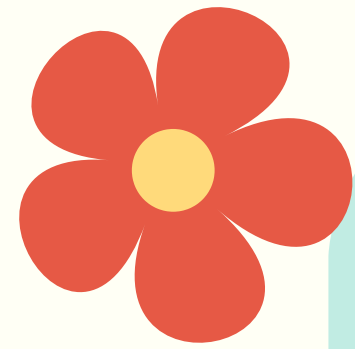
7- Gestion de BDD

8- Explications techniques



Contexte

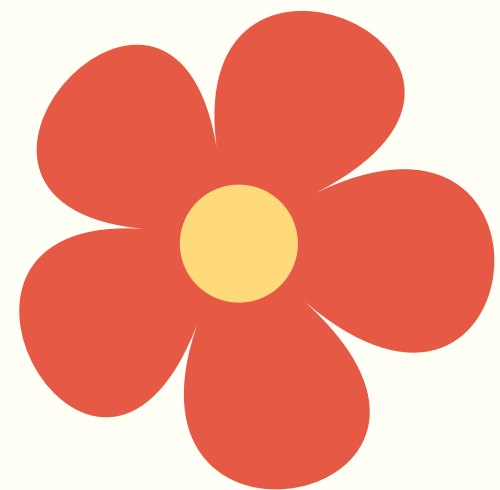
Le but de ce projet était d'effectuer la migration de la BDD existante SQL d'un système scolaire vers une base de No-SQL en utilisant Python comme langage de programmation.



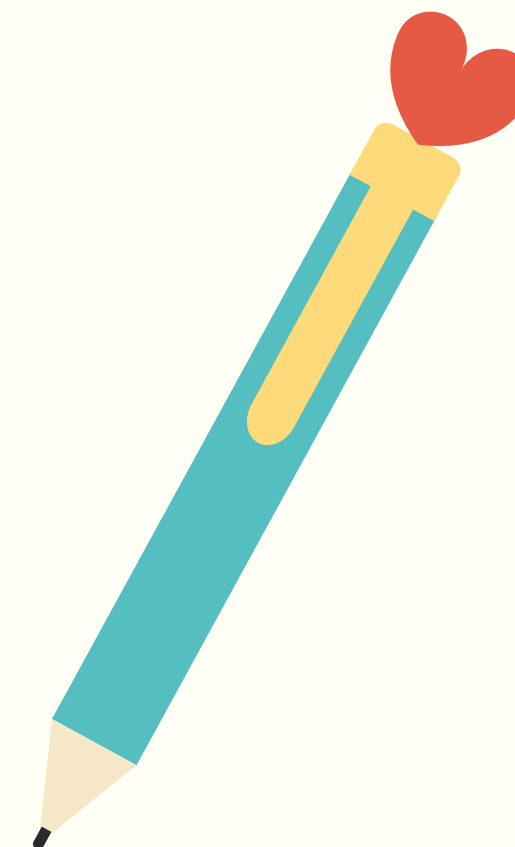
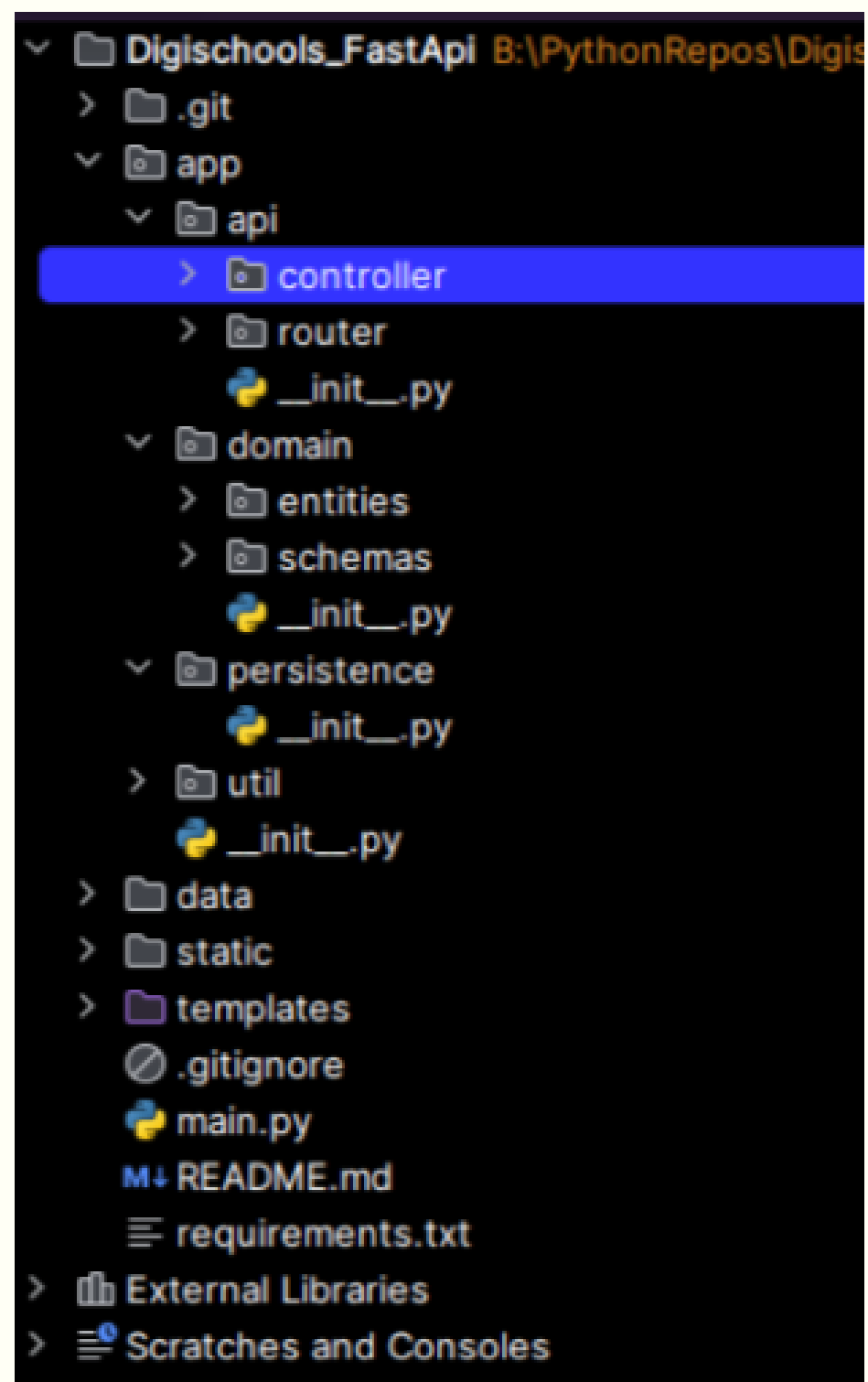
Architecture

Nous avons choisi une architecture "clean" et par couches qui présentent les bénéfices suivants :

- Modularité
- Maintenabilité
- Scalabilité
- Evolutivité



Capture architecture





Migrateur

```
# Connexion à MySQL
mysql_conn = mysql.connector.connect(
    host="localhost",
    port=3306,
    user="digischools",
    password="digischools",
    database="digischools"
)
mysql_cursor = mysql_conn.cursor(dictionary=True)

# Connexion à MongoDB
mongo_db = MongoSingleton.get_db()

# Fonction pour obtenir un enregistrement par ID
1 usage  ↳ Bernardo Estacio Abreu
def get_record_by_id(table, id_field, id_value):
    mysql_cursor.execute(operation=f"SELECT * FROM {table} WHERE {id_field} = %s", params=(id_value,))
    return mysql_cursor.fetchone()

# Fonction pour transformer les enregistrements en remplaçant les ID par les enregistrements complets
1 usage  ↳ Bernardo Estacio Abreu
def transform_record(record, table_structure):
    for field, ref_table in table_structure.items():
        if field in record:
            ref_record = get_record_by_id(ref_table['table'], ref_table['id_field'], record[field])
            if ref_record:
                record[field] = ref_record # Embed the full referenced record instead of the ID
    return record
```

Notre migrateur se divise en 4 étapes fondamentales.

La connexion à la DataBase relationnelle et la recherches des tables présentes.

Cette phase nous permet de reconnaître toutes les tables et leurs ID correspondants aux Objets MongoDB.



Migrateur

Phase de transcription ou traduction :

Cette phase nous permet de définir la bonne structure de collections et de sous-collections MongoDB pour permettre la bonne transition de la table SQL vers No-SQL.



```
# Connexion à MongoDB
mongo_db = MongoSingleton.get_db()

# Fonction pour obtenir un enregistrement par ID
1 usage  ▲ Bernardo Estacio Abreu
def get_record_by_id(table, id_field, id_value):
    mysql_cursor.execute(operation=f"SELECT * FROM {table} WHERE {id_field} = %s", params=(id_value,))
    return mysql_cursor.fetchone()

# Fonction pour transformer les enregistrements en remplaçant les ID par les enregistrements complets
1 usage  ▲ Bernardo Estacio Abreu
def transform_record(record, table_structure):
    for field, ref_table in table_structure.items():
        if field in record:
            ref_record = get_record_by_id(ref_table['table'], ref_table['id_field'], record[field])
            if ref_record:
                record[field] = ref_record # Embed the full referenced record instead of the ID
    return record

# Structure des tables avec les sous-collections
table_structure = {
    "t_notes": {
        "idclasse": {"table": "t_classe", "id_field": "id"},
        "ideleve": {"table": "t_eleve", "id_field": "id"},
        "idmatiere": {"table": "t_matiere", "id_field": "idmatiere"},
        "idprof": {"table": "t_prof", "id_field": "id"},
        "idtrimestre": {"table": "t_trimestre", "id_field": "idtrimestre"}
    },
    "t_classe": {
        "prof": {"table": "t_prof", "id_field": "id"}
    },
    "t_eleve": {
        "classe": {"table": "t_classe", "id_field": "id"}
    },
    "t_prof": {},
    "t_matiere": {},
    "t_trimestre": {}
}
```



Migrateur

```
# Insertion des données dans MongoDB avec des sous-collections
for mysql_table, mongo_collection in tables.items():
    mysql_cursor.execute(f"SELECT * FROM {mysql_table}")
    records = mysql_cursor.fetchall()

    # Si la table a des sous-collections, crée les sous-collections
    if mysql_table in table_structure:
        for record in records:
            transformed_record = transform_record(record, table_structure[mysql_table])
            # Use the primary key as the filter for upsert
            primary_key = list(record.keys())[0]
            # Vérifier si la donnée existe déjà dans la collection pour éviter les doublons
            existing_record = mongo_db[mongo_collection].find_one({primary_key: record[primary_key]})
            if not existing_record:
                mongo_db[mongo_collection].update_one(
                    {primary_key: record[primary_key]},
                    {"$set": transformed_record},
                    upsert=True
                )
    else:
        # Pas de sous-collections, insertion directe
        for record in records:
            primary_key = list(record.keys())[0]
            # Vérifier si la donnée existe déjà dans MongoDB pour éviter les doublons
            existing_record = mongo_db[mongo_collection].find_one({primary_key: record[primary_key]})
            if not existing_record:
                mongo_db[mongo_collection].update_one(
                    {primary_key: record[primary_key]},
                    {"$set": record},
                    upsert=True
                )
```

Phase d'insertion et de vérification des données :

Cette phase nous permet de vérifier la cohérence des données dans les différentes tables, et de les transformer afin de les insérer correctement dans les collections MongoDB.



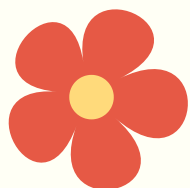
Migrateur

```
# Création des vues dans MongoDB (uniquement après la migration des données)
1 usage  ± Bernardo Estacio Abreu +1
def create_mongo_views():
    # View for Student and Trimester Aggregation
    mongo_db.command({
        'create': 'view_stu_tri',
        'viewOn': 'notes',
        'pipeline': [
            {
                '$unwind': '$ideleve' # Flatten the 'ideleve' field (student info)
            },
            {
                '$unwind': '$idtrimestre' # Flatten the 'idtrimestre' field (trimester info)
            },
            {
                '$group': {
                    '_id': {
                        'eleve_id': '$ideleve.id', # Group by student ID
                        'trimestre_id': '$idtrimestre.idtrimestre' # Group by trimester ID
                    },
                    'eleve_nom': {'$first': '$ideleve.nom'}, # Get student name
                    'eleve_prenom': {'$first': '$ideleve.prenom'}, # Get student first name
                    'classe': {'$first': '$ideleve.classe'}, # Get the student's class
                    'trimestre_nom': {'$first': '$idtrimestre.nom'}, # Get trimester name
                    'trimestre_start': {'$first': '$idtrimestre.date'}, # Get the trimester start date
                    'notes': {'$push': '$note'}, # Collect all the notes related to the student
                    'average_note': {'$avg': '$note'} # Calculate the average note for the student in that trimester
                }
            },
            {
                '$sort': {
                    '_id.eleve_id': 1,
                    '_id.trimestre_id': 1
                }
            },
            {
                '$project': {
                    '_id': 0,
                    'eleve_id': '$_id.eleve_id',
```

Phase d'aggrégation et de création de vues :

Cette phase nous permet de faire les aggrégations correspondant à chaque processus afin de les assigner aux différentes vues pour nous permettre la bonne consultation des données ainsi que leur traitement de manière plus efficace.





Gestion des notes

Dans notre service de gestion de notes, nous avons choisi de créer parallèlement toutes les autres entités reliées aux sous-collections qui prennent en compte la vérification de l'existence de chaque entité pour éviter les doublons.

Cela nous permet de gérer de façon plus efficace la création des données en respectant l'intégrité des données ainsi que la structure des données.

```
# Creation d'une nouvelle note
1 usage  ↳ Bernardo Estacio Abreu +1
def create_note(note: NoteCreateSchema, db: Database):
    # Fonction pour obtenir le prochain ID pour une entité
    ↳ GaelBuenoBarthe +1
    def get_next_id(collection_name: str):
        highest_entry = db[collection_name].find_one(filter={}, sort=[("id", -1)]) # Remplacer "id" par le champ utilisé pour les IDs dans la collection
        if highest_entry and "id" in highest_entry:
            return highest_entry["id"] + 1
        else:
            raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Impossible de déterminer le prochain ID")

    # Vérifier si la classe existe ; créer si elle n'existe pas
    existing_classe = db.classes.find_one({"id": note.idclasse.id})
    if not existing_classe:
        next_classe_id = note.idclasse.id if note.idclasse.id else get_next_id("classes")
        classe_data = {
            "id": next_classe_id,
            "nom": note.idclasse.nom,
            "prof": note.idclasse.prof,
        }
        db.classes.insert_one(classe_data)
        note.idclasse.id = next_classe_id # Mettre à jour la note avec le nouvel ID de la classe
    else:
        note.idclasse.id = existing_classe["id"] # Intégrer l'ID de la classe existante

    # Vérifier si l'élève existe ; créer s'il n'existe pas
    existing_eleve = db.eleves.find_one({"id": note.ideleve.id})
    if not existing_eleve:
        next_eleve_id = note.ideleve.id if note.ideleve.id else get_next_id("eleves")
        eleve_data = {
            "id": next_eleve_id,
            "nom": note.ideleve.nom,
            "prenom": note.ideleve.prenom,
            "classe": note.ideleve.classe,
            "date_naissance": note.ideleve.date_naissance,
            "adresse": note.ideleve.adresse,
            "sexe": note.ideleve.sexe,
        }
        db.eleves.insert_one(eleve_data)
        note.ideleve.id = next_eleve_id
    else:
        note.ideleve.id = existing_eleve["id"] # Intégrer l'ID de l'élève existant

    db.notes.insert_one(note.dict())
```



Technologies

Python 3.12

MongoDB

PyMongo

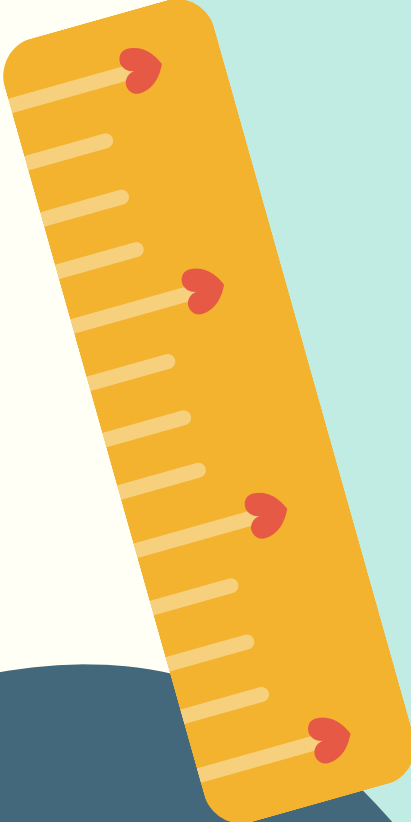
FastAPI

MySQL

Swagger



Fonctionnalités

- Gestion des élèves
 - Gestion professeurs
 - Gestion des notes
 - Gestion des classes
 - Gestion des matières
 - Visualisation des trimestres
- 



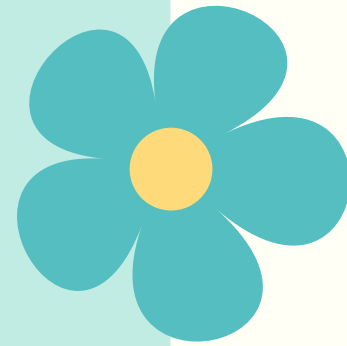
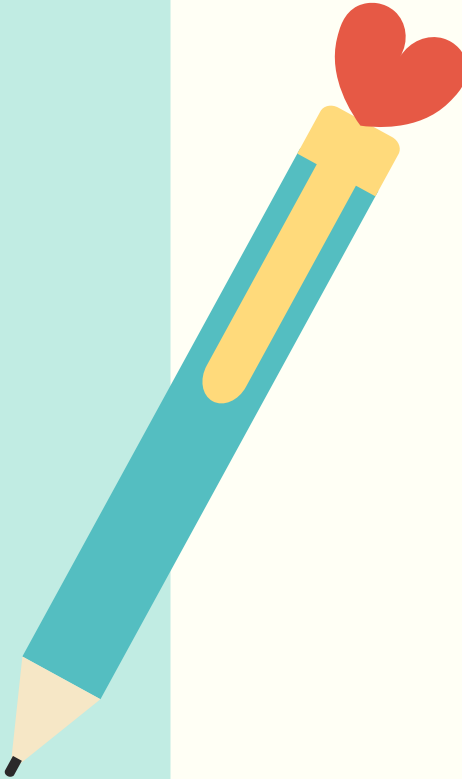

Gestion de BDD

- Création d'objets imbriqués
- Gestion des agrégations
- Création de vues
- Validation des données
- Visualisation/Consultation de vues



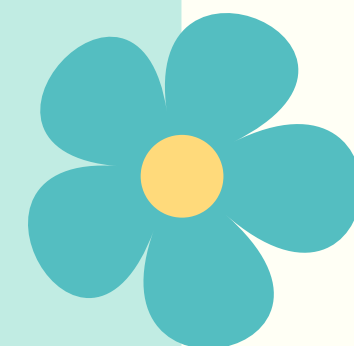
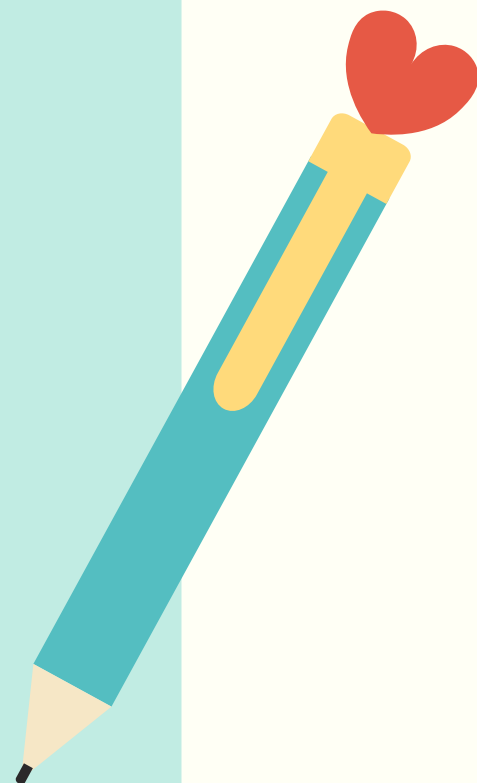
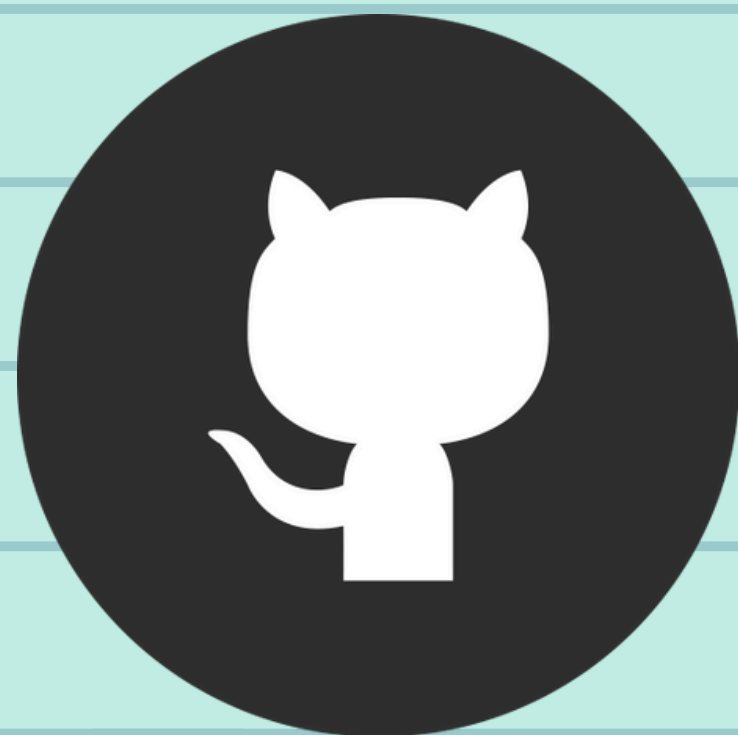
Explications techniques

Nous avons choisi de gérer les aggrégations imbriquées dans des vues respectives afin de permettre une consultation efficace et rapide des données. De la même façon, nous avons dénormalisé la BDD pour permettre une adaptation au modèle No-SQL.





RDU



C'est par ici !