TP Complet GraphQL avec FastAPI, SQLite et Frontend React

Djebabla Ammar

November 27, 2025

Contents

1 Introduction

Ce TP complet présente l'intégration de GraphQL avec FastAPI, SQLite et un frontend React. Nous couvrirons les bonnes pratiques, l'architecture modulaire, et l'intégration complète entre backend et frontend.

2 Architecture du Projet

Structure des fichiers

```
graphql-fastapi-demo/
          backend/
                app/
                      __init__.py
                      main.py
                      database/
                            __init__.py
                            database.py
                            models.py
                      graphql/
                            __init__.py
                            schema.py
                      services/
                          __init__.py
                          user_service.py
                requirements.txt
                scripts/
                    init_data.py
          frontend/
              public/
              src/
                    components/
                          UserList.tsx
                    apollo/
                          client.ts
                    App.tsx
              package.json
```

3 Partie Backend - FastAPI avec GraphQL

3.1 Configuration de la Base de Données

```
# backend/app/database/database.py
import os
```

```python
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

DATABASE_URL = os.getenv("DATABASE_URL", "sqlite:///./test.db")

engine = create_engine(
    DATABASE_URL,
    connect_args={"check_same_thread": False},
    echo=True
)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

3.2 Modèles SQLAlchemy

```python
# backend/app/database/models.py
from sqlalchemy import Column, Integer, String
from .database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    email = Column(String(100), unique=True, index=True, nullable=False)

    def __repr__(self):
        return f"<User(id={self.id}, name='{self.name}', email='{self.email}')>"
```

3.3 Couche Service

```python
# backend/app/services/user_service.py
```

```python
from sqlalchemy.orm import Session
from typing import List, Optional
from app.database.models import User

class UserService:
    @staticmethod
    def get_users(
        db: Session,
        skip: int = 0,
        limit: int = 100,
        name: Optional[str] = None,
        email: Optional[str] = None,
        user_id: Optional[int] = None
    ) -> List[User]:
        query = db.query(User)

        if user_id:
            query = query.filter(User.id == user_id)
        if name:
            query = query.filter(User.name.contains(name))
        if email:
            query = query.filter(User.email.contains(email))

        return query.offset(skip).limit(limit).all()

    @staticmethod
    def create_user(db: Session, name: str, email: str) ->
       User:
        if not name or not email:
            raise ValueError("Name and email are required")

        existing_user = db.query(User).filter(User.email ==
           email).first()
        if existing_user:
            raise ValueError("Email already exists")

        new_user = User(name=name, email=email)
        db.add(new_user)
        db.commit()
        db.refresh(new_user)
        return new_user

    @staticmethod
    def get_user_by_id(db: Session, user_id: int) -> Optional
       [User]:
        return db.query(User).filter(User.id == user_id).
           first()

    @staticmethod
```

```
    def update_user(db: Session, user_id: int, name: str,
        email: str) -> Optional[User]:
        user = db.query(User).filter(User.id == user_id).
            first()
        if user:
            user.name = name
            user.email = email
            db.commit()
            db.refresh(user)
        return user

    @staticmethod
    def delete_user(db: Session, user_id: int) -> bool:
        user = db.query(User).filter(User.id == user_id).
            first()
        if user:
            db.delete(user)
            db.commit()
            return True
        return False
```

3.4 Schema GraphQL

```
# backend/app/graphql/schema.py
import strawberry
from typing import List, Optional
from app.services.user_service import UserService
from app.database.database import get_db

@strawberry.type
class User:
    id: int
    name: str
    email: str

@strawberry.input
class UserInput:
    name: str
    email: str

@strawberry.input
class UserFilter:
    id: Optional[int] = None
    name: Optional[str] = None
    email: Optional[str] = None

@strawberry.type
```

```python
class Query:
    @strawberry.field
    def users(
        self,
        filter: Optional[UserFilter] = None,
        skip: int = 0,
        limit: int = 100
    ) -> List[User]:
        db = next(get_db())

        filter_dict = {}
        if filter:
            filter_dict = {
                'user_id': filter.id,
                'name': filter.name,
                'email': filter.email
            }

        users = UserService.get_users(
            db=db,
            skip=skip,
            limit=limit,
            **{k: v for k, v in filter_dict.items() if v is not None}
        )

        return [
            User(id=user.id, name=user.name, email=user.email)
            for user in users
        ]

    @strawberry.field
    def user(self, id: int) -> Optional[User]:
        db = next(get_db())
        user = UserService.get_user_by_id(db, id)
        if user:
            return User(id=user.id, name=user.name, email=user.email)
        return None

@strawberry.type
class Mutation:
    @strawberry.mutation
    def create_user(self, user_input: UserInput) -> User:
        db = next(get_db())
        try:
            new_user = UserService.create_user(
                db=db,
```

```
                name=user_input.name,
                email=user_input.email
            )
            return User(
                id=new_user.id,
                name=new_user.name,
                email=new_user.email
            )
        except ValueError as e:
            raise Exception(str(e))

    @strawberry.mutation
    def update_user(self, id: int, user_input: UserInput) ->
        Optional[User]:
        db = next(get_db())
        user = UserService.update_user(db, id, user_input.
            name, user_input.email)
        if user:
            return User(id=user.id, name=user.name, email=
                user.email)
        return None

    @strawberry.mutation
    def delete_user(self, id: int) -> bool:
        db = next(get_db())
        return UserService.delete_user(db, id)

schema = strawberry.Schema(query=Query, mutation=Mutation)
```

3.5 Application FastAPI Principale

```
# backend/app/main.py
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from strawberry.fastapi import GraphQLRouter
from .graphql.schema import schema

def create_application() -> FastAPI:
    app = FastAPI(
        title="GraphQL FastAPI Demo",
        description="API GraphQL avec FastAPI et SQLite",
        version="1.0.0"
    )

    # Configuration CORS
    app.add_middleware(
        CORSMiddleware,
```

```
        allow_origins=["http://localhost:3000", "http
            ://127.0.0.1:3000"],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )

    # Route GraphQL
    graphql_app = GraphQLRouter(schema)
    app.include_router(graphql_app, prefix="/graphql")

    # Routes sant
    @app.get("/")
    async def root():
        return {"message": "GraphQL API is running"}

    @app.get("/health")
    async def health_check():
        return {"status": "healthy"}

    return app

app = create_application()
```

3.6 Script d'Initialisation des Données

```
# backend/scripts/init_data.py
import sys
import os
sys.path.append(os.path.dirname(os.path.dirname(os.path.
    abspath(__file__))))

from app.database.database import engine, SessionLocal
from app.database.models import User

def init_database():
    # Cr er les tables
    User.metadata.create_all(bind=engine)

    db = SessionLocal()

    # V rifier si des donn es existent d j
    if db.query(User).count() == 0:
        users = [
            User(name="Alice Dupont", email="alice@example.
                com"),
            User(name="Bob Martin", email="bob@example.com"),
```

```
            User(name="Charlie Brown", email="charlie@example.com"),
        ]

        db.add_all(users)
        db.commit()
        print("✅ Données initiales créées avec succès!")
    else:
        print("ℹ️  La base contient déjà des données.")

    db.close()

if __name__ == "__main__":
    init_database()
```

3.7 Fichier Requirements

```
# backend/requirements.txt
fastapi==0.104.0
uvicorn[standard]==0.23.2
strawberry-graphql==1.12.2
sqlalchemy==2.1.2
python-multipart==0.0.6
pydantic==2.5.0
```

4 Partie Frontend - React avec Apollo Client

4.1 Configuration Apollo Client

```
// frontend/src/apollo/client.ts
import { ApolloClient, InMemoryCache, createHttpLink } from '@apollo/client';

const httpLink = createHttpLink({
  uri: 'http://localhost:8000/graphql',
});

export const client = new ApolloClient({
  link: httpLink,
  cache: new InMemoryCache(),
  defaultOptions: {
    watchQuery: {
```

```
      fetchPolicy: 'cache-and-network',
    },
  },
});
```

4.2 Composant Principal de l'Application

```
// frontend/src/App.tsx
import React from 'react';
import { ApolloProvider } from '@apollo/client';
import { client } from './apollo/client';
import { UserList } from './components/UserList';
import './App.css';

function App() {
  return (
    <ApolloProvider client={client}>
      <div className="App">
        <header className="App-header">
          <h1>GraphQL FastAPI Demo</h1>
        </header>
        <main>
          <UserList />
        </main>
      </div>
    </ApolloProvider>
  );
}

export default App;
```

4.3 Composant de Gestion des Utilisateurs

```
// frontend/src/components/UserList.tsx
import React, { useState } from 'react';
import { useQuery, useMutation, gql } from '@apollo/client';

const GET_USERS = gql`
  query GetUsers($filter: UserFilter, $skip: Int, $limit: Int
     ) {
    users(filter: $filter, skip: $skip, limit: $limit) {
      id
      name
      email
    }
```

```
  }
`;

const CREATE_USER = gql`
  mutation CreateUser($userInput: UserInput!) {
    createUser(userInput: $userInput) {
      id
      name
      email
    }
  }
`;

const UPDATE_USER = gql`
  mutation UpdateUser($id: Int!, $userInput: UserInput!) {
    updateUser(id: $id, userInput: $userInput) {
      id
      name
      email
    }
  }
`;

const DELETE_USER = gql`
  mutation DeleteUser($id: Int!) {
    deleteUser(id: $id)
  }
`;

export const UserList: React.FC = () => {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [searchTerm, setSearchTerm] = useState('');
  const [editingUser, setEditingUser] = useState<any>(null);

  const { loading, error, data, refetch } = useQuery(
     GET_USERS, {
    variables: {
      filter: searchTerm ? { name: searchTerm } : undefined,
      limit: 50
    }
  });

  const [createUser] = useMutation(CREATE_USER, {
    onCompleted: () => {
      setName('');
      setEmail('');
      refetch();
    }
```

```
  });

  const [updateUser] = useMutation(UPDATE_USER, {
    onCompleted: () => {
      setEditingUser(null);
      refetch();
    }
  });

  const [deleteUser] = useMutation(DELETE_USER, {
    onCompleted: () => refetch()
  });

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (editingUser) {
      updateUser({
        variables: {
          id: editingUser.id,
          userInput: { name, email }
        }
      });
    } else {
      createUser({
        variables: {
          userInput: { name, email }
        }
      });
    }
  };

  const handleEdit = (user: any) => {
    setEditingUser(user);
    setName(user.name);
    setEmail(user.email);
  };

  const handleCancel = () => {
    setEditingUser(null);
    setName('');
    setEmail('');
  };

  const handleSearch = (e: React.FormEvent) => {
    e.preventDefault();
    refetch({
      filter: searchTerm ? { name: searchTerm } : undefined
    });
  };
```

```
109
110   if (loading) return <div className="loading">Chargement
          ...</div>;
111   if (error) return <div className="error">Erreur: {error.
          message}</div>;
112
113   return (
114     <div className="user-management">
115       <h2>Gestion des Utilisateurs</h2>
116
117       {/* Formulaire de recherche */}
118       <form onSubmit={handleSearch} className="search-form">
119         <input
120           type="text"
121           placeholder="Rechercher par nom..."
122           value={searchTerm}
123           onChange={(e) => setSearchTerm(e.target.value)}
124         />
125         <button type="submit">Rechercher</button>
126         <button
127           type="button"
128           onClick={() => {
129             setSearchTerm('');
130             refetch({ filter: undefined });
131           }}
132         >
133           Effacer
134         </button>
135       </form>
136
137       {/* Formulaire de cr ation/ dition  */}
138       <form onSubmit={handleSubmit} className="user-form">
139         <h3>{editingUser ? 'Modifier' : 'Ajouter'} un
              utilisateur</h3>
140         <div className="form-group">
141           <input
142             type="text"
143             placeholder="Nom"
144             value={name}
145             onChange={(e) => setName(e.target.value)}
146             required
147           />
148         </div>
149         <div className="form-group">
150           <input
151             type="email"
152             placeholder="Email"
153             value={email}
154             onChange={(e) => setEmail(e.target.value)}
```

```
            required
          />
        </div>
        <div className="form-actions">
          <button type="submit">
            {editingUser ? 'Modifier' : 'Cr er'} Utilisateur
          </button>
          {editingUser && (
            <button type="button" onClick={handleCancel}>
              Annuler
            </button>
          )}
        </div>
      </form>

      {/* Liste des utilisateurs */}
      <div className="user-list">
        <h3>Liste des Utilisateurs ({data?.users.length})</h3
          >
        {data?.users.map((user: any) => (
          <div key={user.id} className="user-card">
            <div className="user-info">
              <strong>{user.name}</strong>
              <span className="user-email">{user.email}</span
                >
            </div>
            <div className="user-actions">
              <button onClick={() => handleEdit(user)}>
                Modifier
              </button>
              <button
                onClick={() => {
                  if (window.confirm('Supprimer cet
                     utilisateur ?')) {
                    deleteUser({ variables: { id: user.id }
                       });
                  }
                }}
                className="delete-btn"
              >
                Supprimer
              </button>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};
```

4.4 Styles CSS

```css
/* frontend/src/App.css */
.App {
  text-align: center;
}

.App-header {
  background-color: #282c34;
  padding: 20px;
  color: white;
}

.user-management {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
}

.search-form, .user-form {
  background: #f5f5f5;
  padding: 20px;
  margin: 20px 0;
  border-radius: 8px;
}

.form-group {
  margin: 10px 0;
}

.form-group input {
  width: 100%;
  padding: 8px;
  margin: 5px 0;
  border: 1px solid #ddd;
  border-radius: 4px;
}

.form-actions {
  display: flex;
  gap: 10px;
  margin-top: 15px;
}

button {
  padding: 8px 16px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
```

```
  background-color: #007bff;
  color: white;
}

button:hover {
  background-color: #0056b3;
}

.delete-btn {
  background-color: #dc3545;
}

.delete-btn:hover {
  background-color: #c82333;
}

.user-list {
  margin-top: 30px;
}

.user-card {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 15px;
  margin: 10px 0;
  border: 1px solid #ddd;
  border-radius: 8px;
  background: white;
}

.user-info {
  text-align: left;
}

.user-email {
  display: block;
  color: #666;
  font-size: 0.9em;
}

.user-actions {
  display: flex;
  gap: 10px;
}

.loading {
  padding: 20px;
  font-size: 1.2em;
```

```
}

.error {
  padding: 20px;
  color: #dc3545;
  background: #f8d7da;
  border: 1px solid #f5c6cb;
  border-radius: 4px;
}
```

4.5 Package.json Frontend

```
{
  "name": "graphql-frontend",
  "version": "1.0.0",
  "dependencies": {
    "@apollo/client": "^3.8.0",
    "graphql": "^16.8.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

5 Guide d'Exécution

5.1 Lancement du Backend

```
# Terminal 1 - Backend
cd backend
python -m venv .venv
source .venv/bin/activate  # Linux/Mac
# .venv\Scripts\activate   # Windows

pip install -r requirements.txt
python scripts/init_data.py
uvicorn app.main:app --reload --port 8000
```

5.2 Lancement du Frontend

```
# Terminal 2 - Frontend
cd frontend
npm install
npm start
```

5.3 Accès aux Applications

- **Backend API**: `http://localhost:8000`
- **GraphQL Playground**: `http://localhost:8000/graphql`
- **Frontend**: `http://localhost:3000`

6 Exemples de Requêtes GraphQL

6.1 Queries

```
# R cup rer tous les utilisateurs
query {
  users {
    id
    name
    email
  }
}

# Rechercher par nom
```

```
query {
  users(filter: { name: "Alice" }) {
    id
    name
    email
  }
}

# R cup rer un utilisateur sp cifique
query {
  user(id: 1) {
    id
    name
    email
  }
}
```

6.2 Mutations

```
# Cr er un utilisateur
mutation {
  createUser(userInput: {
    name: "David Wilson",
    email: "david@example.com"
  }) {
    id
    name
    email
  }
}

# Modifier un utilisateur
mutation {
  updateUser(
    id: 1,
    userInput: {
      name: "Alice Smith",
      email: "alice.smith@example.com"
    }
  ) {
    id
    name
    email
  }
}

# Supprimer un utilisateur
```

```
mutation {
  deleteUser(id: 3)
}
```

7 Conclusion

Ce TP complet a couvert :

- **Architecture modulaire** avec séparation des concerns
- **GraphQL** avec Strawberry pour le schéma et les résolveurs
- **FastAPI** avec configuration CORS pour le backend
- **SQLAlchemy** pour l'ORM et la gestion de la base SQLite
- **Apollo Client** pour l'intégration frontend React
- **Bonnes pratiques** : services layer, validation, gestion d'erreurs

L'application démontre un CRUD complet avec recherche, création, modification et suppression d'utilisateurs, le tout avec une architecture moderne et maintenable.