

TP Complet GraphQL avec FastAPI, SQLite et Frontend React

Djebabla Ammar

November 27, 2025

Contents

1	Introduction	3
2	Architecture du Projet	3
3	Partie Backend - FastAPI avec GraphQL	3
3.1	Configuration de la Base de Données	3
3.2	Modèles SQLAlchemy	4
3.3	Couche Service	4
3.4	Schema GraphQL	6
3.5	Application FastAPI Principale	8
3.6	Script d'Initialisation des Données	9
3.7	Fichier Requirements	10
4	Partie Frontend - React avec Apollo Client	10
4.1	Configuration Apollo Client	10
4.2	Composant Principal de l'Application	11
4.3	Composant de Gestion des Utilisateurs	11
4.4	Styles CSS	16
4.5	Package.json Frontend	18
5	Guide d'Exécution	19
5.1	Lancement du Backend	19
5.2	Lancement du Frontend	19
5.3	Accès aux Applications	19

6 Exemples de Requêtes GraphQL	19
6.1 Queries	19
6.2 Mutations	20
7 Conclusion	21

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

```
1 graphql-fastapi-demo/
2     backend/
3         app/
4             __init__.py
5             main.py
6             database/
7                 __init__.py
8                 database.py
9                 models.py
10            graphql/
11                __init__.py
12                schema.py
13                services/
14                    __init__.py
15                    user_service.py
16                requirements.txt
17                scripts/
18                    init_data.py
19            frontend/
20                public/
21                src/
22                    components/
23                        UserList.tsx
24                    apollo/
25                        client.ts
26                    App.tsx
27                    package.json
```

3 Partie Backend - FastAPI avec GraphQL

3.1 Configuration de la Base de Données

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

```

3 | from sqlalchemy import create_engine
4 | from sqlalchemy.ext.declarative import declarative_base
5 | from sqlalchemy.orm import sessionmaker
6 |
7 | DATABASE_URL = os.getenv("DATABASE_URL", "sqlite:///./test.db")
8 |
9 | engine = create_engine(
10 |     DATABASE_URL,
11 |     connect_args={"check_same_thread": False},
12 |     echo=True
13 | )
14 |
15 | SessionLocal = sessionmaker(autocommit=False, autoflush=False
16 |     , bind=engine)
16 | Base = declarative_base()
17 |
18 | def get_db():
19 |     db = SessionLocal()
20 |     try:
21 |         yield db
22 |     finally:
23 |         db.close()

```

3.2 Modèles SQLAlchemy

```

1 # backend/app/database/models.py
2 from sqlalchemy import Column, Integer, String
3 from .database import Base
4
5 class User(Base):
6     __tablename__ = "users"
7
8     id = Column(Integer, primary_key=True, index=True)
9     name = Column(String(100), nullable=False)
10    email = Column(String(100), unique=True, index=True,
11                  nullable=False)
12
13    def __repr__(self):
14        return f"<User(id={self.id}, name='{self.name}', "
15                    email='{self.email}')>"

```

3.3 Couche Service

```

1 # backend/app/services/user_service.py

```

```

2 | from sqlalchemy.orm import Session
3 | from typing import List, Optional
4 | from app.database.models import User
5 |
6 | class UserService:
7 |     @staticmethod
8 |     def get_users(
9 |         db: Session,
10 |         skip: int = 0,
11 |         limit: int = 100,
12 |         name: Optional[str] = None,
13 |         email: Optional[str] = None,
14 |         user_id: Optional[int] = None
15 |     ) -> List[User]:
16 |         query = db.query(User)
17 |
18 |         if user_id:
19 |             query = query.filter(User.id == user_id)
20 |         if name:
21 |             query = query.filter(User.name.contains(name))
22 |         if email:
23 |             query = query.filter(User.email.contains(email))
24 |
25 |         return query.offset(skip).limit(limit).all()
26 |
27 |     @staticmethod
28 |     def create_user(db: Session, name: str, email: str) ->
29 |         User:
30 |         if not name or not email:
31 |             raise ValueError("Name and email are required")
32 |
33 |         existing_user = db.query(User).filter(User.email ==
34 |             email).first()
35 |         if existing_user:
36 |             raise ValueError("Email already exists")
37 |
38 |         new_user = User(name=name, email=email)
39 |         db.add(new_user)
40 |         db.commit()
41 |         db.refresh(new_user)
42 |         return new_user
43 |
44 |     @staticmethod
45 |     def get_user_by_id(db: Session, user_id: int) -> Optional
46 |         [User]:
47 |             return db.query(User).filter(User.id == user_id).
48 |                 first()
49 |
50 |     @staticmethod

```

```

47     def update_user(db: Session, user_id: int, name: str,
48                     email: str) -> Optional[User]:
49         user = db.query(User).filter(User.id == user_id).
50             first()
51         if user:
52             user.name = name
53             user.email = email
54             db.commit()
55             db.refresh(user)
56         return user
57
58     @staticmethod
59     def delete_user(db: Session, user_id: int) -> bool:
60         user = db.query(User).filter(User.id == user_id).
61             first()
62         if user:
63             db.delete(user)
64             db.commit()
65             return True
66         return False

```

3.4 Schema GraphQL

```

1 # backend/app/graphql/schema.py
2 import strawberry
3 from typing import List, Optional
4 from app.services.user_service import UserService
5 from app.database.database import get_db
6
7 @strawberry.type
8 class User:
9     id: int
10    name: str
11    email: str
12
13 @strawberry.input
14 class UserInput:
15     name: str
16     email: str
17
18 @strawberry.input
19 class UserFilter:
20     id: Optional[int] = None
21     name: Optional[str] = None
22     email: Optional[str] = None
23
24 @strawberry.type

```

```

25 | class Query:
26 |     @strawberry.field
27 |     def users(
28 |         self,
29 |         filter: Optional[UserFilter] = None,
30 |         skip: int = 0,
31 |         limit: int = 100
32 |     ) -> List[User]:
33 |         db = next(get_db())
34 |
35 |         filter_dict = {}
36 |         if filter:
37 |             filter_dict = {
38 |                 'user_id': filter.id,
39 |                 'name': filter.name,
40 |                 'email': filter.email
41 |             }
42 |
43 |         users = UserService.get_users(
44 |             db=db,
45 |             skip=skip,
46 |             limit=limit,
47 |             **{k: v for k, v in filter_dict.items() if v is
48 |                 not None}
49 |         )
50 |
51 |         return [
52 |             User(id=user.id, name=user.name, email=user.email
53 |                  )
54 |             for user in users
55 |         ]
56 |
57 |     @strawberry.field
58 |     def user(self, id: int) -> Optional[User]:
59 |         db = next(get_db())
60 |         user = UserService.get_user_by_id(db, id)
61 |         if user:
62 |             return User(id=user.id, name=user.name, email=
63 |                         user.email)
64 |         return None
65 |
66 | @strawberry.type
67 | class Mutation:
68 |     @strawberry.mutation
69 |     def create_user(self, user_input: UserInput) -> User:
70 |         db = next(get_db())

```

```

71             name=user_input.name,
72             email=user_input.email
73         )
74     return User(
75         id=new_user.id,
76         name=new_user.name,
77         email=new_user.email
78     )
79 except ValueError as e:
80     raise Exception(str(e))
81
82 @strawberry.mutation
83 def update_user(self, id: int, user_input: UserInput) ->
84     Optional[User]:
85     db = next(get_db())
86     user = UserService.update_user(db, id, user_input.
87         name, user_input.email)
88     if user:
89         return User(id=user.id, name=user.name, email=
90             user.email)
91     return None
92
93 @strawberry.mutation
94 def delete_user(self, id: int) -> bool:
95     db = next(get_db())
96     return UserService.delete_user(db, id)
97
98 schema = strawberry.Schema(query=Query, mutation=Mutation)

```

3.5 Application FastAPI Principale

```

1 # backend/app/main.py
2 from fastapi import FastAPI
3 from fastapi.middleware.cors import CORSMiddleware
4 from strawberry.fastapi import GraphQLRouter
5 from .graphql.schema import schema
6
7 def create_application() -> FastAPI:
8     app = FastAPI(
9         title="GraphQL FastAPI Demo",
10        description="API GraphQL avec FastAPI et SQLite",
11        version="1.0.0"
12    )
13
14    # Configuration CORS
15    app.add_middleware(
16        CORSMiddleware,

```

```
17     allow_origins=["http://localhost:3000", "http
18                 ://127.0.0.1:3000"],
19     allow_credentials=True,
20     allow_methods=["*"],
21     allow_headers=["*"],
22 )
23
24 # Route GraphQL
25 graphql_app = GraphQLRouter(schema)
26 app.include_router(graphql_app, prefix="/graphql")
27
28 # Routes sant
29 @app.get("/")
30 async def root():
31     return {"message": "GraphQL API is running"}
32
33 @app.get("/health")
34 async def health_check():
35     return {"status": "healthy"}
36
37
38 return app

app = create_application()
```

3.6 Script d'Initialisation des Données

```
1 # backend/scripts/init_data.py
2 import sys
3 import os
4 sys.path.append(os.path.dirname(os.path.dirname(os.path.
5     abspath(__file__))))
6
7 from app.database.database import engine, SessionLocal
8 from app.database.models import User
9
10 def init_database():
11     # Cr er les tables
12     User.metadata.create_all(bind=engine)
13
14     db = SessionLocal()
15
16     # V rifier si des donn es existent d j
17     if db.query(User).count() == 0:
18         users = [
19             User(name="Alice Dupont", email="alice@example.
20                 com"),
21             User(name="Bob Martin", email="bob@example.com"),
```

```

20         User(name="Charlie Brown", email="charlie@example
21             .com"),
22     ]
23
24     db.add_all(users)
25     db.commit()
26     print("      Données initiales créées avec succès!")
27     )
28 else:
29     print("      La base contient déjà des données."
30
31 db.close()
32
33 if __name__ == "__main__":
34     init_database()

```

3.7 Fichier Requirements

```

1 # backend/requirements.txt
2 fastapi==0.104.0
3 uvicorn[standard]==0.23.2
4 strawberry-graphql==1.12.2
5 sqlalchemy==2.1.2
6 python-multipart==0.0.6
7 pydantic==2.5.0

```

4 Partie Frontend - React avec Apollo Client

4.1 Configuration Apollo Client

```

1 // frontend/src/apollo/client.ts
2 import { ApolloClient, InMemoryCache, createHttpLink } from '@apollo/client';
3
4 const httpLink = createHttpLink({
5   uri: 'http://localhost:8000/graphql',
6 });
7
8 export const client = new ApolloClient({
9   link: httpLink,
10   cache: new InMemoryCache(),
11   defaultOptions: {
12     watchQuery: {

```

```

13     fetchPolicy: 'cache-and-network',
14   },
15 },
16 );

```

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
    }
  }

```

```

12     }
13   ';
14
15 const CREATE_USER = gql`  

16   mutation CreateUser($userInput: UserInput!) {  

17     createUser(userInput: $userInput) {  

18       id  

19       name  

20       email  

21     }  

22   }  

23 `;
24
25 const UPDATE_USER = gql`  

26   mutation UpdateUser($id: Int!, $userInput: UserInput!) {  

27     updateUser(id: $id, userInput: $userInput) {  

28       id  

29       name  

30       email  

31     }  

32   }  

33 `;
34
35 const DELETE_USER = gql`  

36   mutation DeleteUser($id: Int!) {  

37     deleteUser(id: $id)  

38   }  

39 `;
40
41 export const UserList: React.FC = () => {
42   const [name, setName] = useState('');
43   const [email, setEmail] = useState('');
44   const [searchTerm, setSearchTerm] = useState('');
45   const [editingUser, setEditingUser] = useState<any>(null);
46
47   const { loading, error, data, refetch } = useQuery(
48     GET_USERS, {
49       variables: {
50         filter: searchTerm ? { name: searchTerm } : undefined,
51         limit: 50
52       }
53     });
54
55   const [createUser] = useMutation(CREATE_USER, {
56     onCompleted: () => {
57       setName('');
58       setEmail('');
59       refetch();
60     }
61   );
62
63   return (
64     <div>
65       <h1>User List</h1>
66       <table>
67         <thead>
68           <tr>
69             <th>Name</th>
70             <th>Email</th>
71             <th>Actions</th>
72           </tr>
73         </thead>
74         <tbody>
75           {data?.users.map((user) => (
76             <tr key={user.id}>
77               <td>{user.name}</td>
78               <td>{user.email}</td>
79               <td>
80                 <button onClick={() => setEditingUser(user)}>Edit</button>
81                 <button onClick={() => deleteUser(user.id)}>Delete</button>
82               </td>
83             </tr>
84           ))}
85         </tbody>
86       </table>
87       <form>
88         <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
89         <input type="text" value={email} onChange={(e) => setEmail(e.target.value)} />
90         <button onClick={() => createUser()}>Create</button>
91       </form>
92     </div>
93   );
94 }

```

```

60   });
61
62   const [updateUser] = useMutation(UPDATE_USER, {
63     onCompleted: () => {
64       setEditingUser(null);
65       refetch();
66     }
67   });
68
69   const [deleteUser] = useMutation(DELETE_USER, {
70     onCompleted: () => refetch()
71   });
72
73   const handleSubmit = (e: React.FormEvent) => {
74     e.preventDefault();
75     if (editingUser) {
76       updateUser({
77         variables: {
78           id: editingUser.id,
79           userInput: { name, email }
80         }
81       });
82     } else {
83       createUser({
84         variables: {
85           userInput: { name, email }
86         }
87       });
88     }
89   };
90
91   const handleEdit = (user: any) => {
92     setEditingUser(user);
93     setName(user.name);
94     setEmail(user.email);
95   };
96
97   const handleCancel = () => {
98     setEditingUser(null);
99     setName('');
100    setEmail('');
101  };
102
103  const handleSearch = (e: React.FormEvent) => {
104    e.preventDefault();
105    refetch({
106      filter: searchTerm ? { name: searchTerm } : undefined
107    });
108  };

```

```

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

```

155     required
156   />
157 </div>
158 <div className="form-actions">
159   <button type="submit">
160     {editingUser ? 'Modifier' : 'Cr er'} Utilisateur
161   </button>
162   {editingUser && (
163     <button type="button" onClick={handleCancel}>
164       Annuler
165     </button>
166   )}
167 </div>
168 </form>
169
170 /* Liste des utilisateurs */
171 <div className="user-list">
172   <h3>Liste des Utilisateurs ({data?.users.length})</h3>
173   >
174   {data?.users.map((user: any) => (
175     <div key={user.id} className="user-card">
176       <div className="user-info">
177         <strong>{user.name}</strong>
178         <span className="user-email">{user.email}</span>
179       </div>
180       <div className="user-actions">
181         <button onClick={() => handleEdit(user)}>
182           Modifier
183         </button>
184         <button
185           onClick={() => {
186             if (window.confirm('Supprimer cet
187               utilisateur ?')) {
188               deleteUser({ variables: { id: user.id } });
189             }
190           }
191         }
192         <span className="delete-btn">
193           Supprimer
194         </span>
195       </div>
196     ))}
197   </div>
198 );
199 >;

```

4.4 Styles CSS

```
1  /* frontend/src/App.css */
2  .App {
3      text-align: center;
4  }
5
6  .App-header {
7      background-color: #282c34;
8      padding: 20px;
9      color: white;
10 }
11
12 .user-management {
13     max-width: 800px;
14     margin: 0 auto;
15     padding: 20px;
16 }
17
18 .search-form, .user-form {
19     background: #f5f5f5;
20     padding: 20px;
21     margin: 20px 0;
22     border-radius: 8px;
23 }
24
25 .form-group {
26     margin: 10px 0;
27 }
28
29 .form-group input {
30     width: 100%;
31     padding: 8px;
32     margin: 5px 0;
33     border: 1px solid #ddd;
34     border-radius: 4px;
35 }
36
37 .form-actions {
38     display: flex;
39     gap: 10px;
40     margin-top: 15px;
41 }
42
43 button {
44     padding: 8px 16px;
45     border: none;
46     border-radius: 4px;
47     cursor: pointer;
```

```
48     background-color: #007bff;
49     color: white;
50 }
51
52 button:hover {
53     background-color: #0056b3;
54 }
55
56 .delete-btn {
57     background-color: #dc3545;
58 }
59
60 .delete-btn:hover {
61     background-color: #c82333;
62 }
63
64 .user-list {
65     margin-top: 30px;
66 }
67
68 .user-card {
69     display: flex;
70     justify-content: space-between;
71     align-items: center;
72     padding: 15px;
73     margin: 10px 0;
74     border: 1px solid #ddd;
75     border-radius: 8px;
76     background: white;
77 }
78
79 .user-info {
80     text-align: left;
81 }
82
83 .user-email {
84     display: block;
85     color: #666;
86     font-size: 0.9em;
87 }
88
89 .user-actions {
90     display: flex;
91     gap: 10px;
92 }
93
94 .loading {
95     padding: 20px;
96     font-size: 1.2em;
```

```
97 | }
98 |
99 | .error {
100|   padding: 20px;
101|   color: #dc3545;
102|   background: #f8d7da;
103|   border: 1px solid #f5c6cb;
104|   border-radius: 4px;
105| }
```

4.5 Package.json Frontend

```
1 {
2   "name": "graphql-frontend",
3   "version": "1.0.0",
4   "dependencies": {
5     "@apollo/client": "^3.8.0",
6     "graphql": "^16.8.0",
7     "react": "^18.2.0",
8     "react-dom": "^18.2.0",
9     "react-scripts": "5.0.1"
10 },
11   "scripts": {
12     "start": "react-scripts start",
13     "build": "react-scripts build",
14     "test": "react-scripts test",
15     "eject": "react-scripts eject"
16   },
17   "browserslist": {
18     "production": [
19       ">0.2%",
20       "not dead",
21       "not op_mini all"
22     ],
23     "development": [
24       "last 1 chrome version",
25       "last 1 firefox version",
26       "last 1 safari version"
27     ]
28   }
29 }
```

5 Guide d'Exécution

5.1 Lancement du Backend

```
1 # Terminal 1 - Backend
2 cd backend
3 python -m venv .venv
4 source .venv/bin/activate # Linux/Mac
5 # .venv\Scripts\activate # Windows
6
7 pip install -r requirements.txt
8 python scripts/init_data.py
9 uvicorn app.main:app --reload --port 8000
```

5.2 Lancement du Frontend

```
1 # Terminal 2 - Frontend
2 cd frontend
3 npm install
4 npm start
```

5.3 Accès aux Applications

- Backend API: <http://localhost:8000>
- GraphQL Playground: <http://localhost:8000/graphq>
- Frontend: <http://localhost:3000>

6 Exemples de Requêtes GraphQL

6.1 Queries

```
1 # Récupérer tous les utilisateurs
2 query {
3   users {
4     id
5     name
6     email
7   }
8 }
9
10 # Rechercher par nom
```

```

11 query {
12   users(filter: { name: "Alice" }) {
13     id
14     name
15     email
16   }
17 }
18
19 # Récupérer un utilisateur spécifique
20 query {
21   user(id: 1) {
22     id
23     name
24     email
25   }
26 }
```

6.2 Mutations

```

1 # Créer un utilisateur
2 mutation {
3   createUser(userInput: {
4     name: "David Wilson",
5     email: "david@example.com"
6   }) {
7     id
8     name
9     email
10   }
11 }
12
13 # Modifier un utilisateur
14 mutation {
15   updateUser(
16     id: 1,
17     userInput: {
18       name: "Alice Smith",
19       email: "alice.smith@example.com"
20     }
21   ) {
22     id
23     name
24     email
25   }
26 }
27
28 # Supprimer un utilisateur
```

```
29 | mutation {
30 |   deleteUser(id: 3)
31 | }
```

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.