GraphQL Integration

Official documentation:

https://graphql.org/learn/

Comparison of main characteristics with REST API.

	REST API	**GraphQL**
Query flexibility	Fixed endpoints and responses	Flexible queries, only the required data is returned
Client performance	Multiple requests needed for different resources	Data from multiple sources can be queried in a single request
Data	Multiple queries may be needed for complex data	All required data can be fetched in a single request
Network efficiency	Network traffic increases with multiple requests	Reduces unnecessary data transfer and number of queries
Usability	Simple, widely known	More difficult to learn, but more efficient
Caching	HTTP cache and CDN supported	Harder to implement, requires custom caching strategy
Versioning	API versioning needed (e.g. `/v1/users`)	No versioning needed, the client selects the required fields
Handling schema changes	Changes may require new endpoints	New fields can be added while keeping the old ones
Security and access control	Built-in HTTP security, roles	Fine-grained access rules needed (e.g., field-level permissions)
Supported formats	Usually JSON (but others supported)	JSON-based (strictly follows GraphQL schema)
Use cases	Ideal for simple CRUD APIs	Better choice for complex, dynamic client needs

After creating a Python virtual environment, you can install the required dependencies with the following command:

```
pip install fastapi strawberry-graphql uvicorn
```

Sample code for first use:

```
import strawberry
from fastapi import FastAPI
from strawberry.fastapi import GraphQLRouter

# User data model
@strawberry.type
class User:
    id: int
    name: str
    age: int
```

```
# Example database
users = [
    User(id=1, name="Noa", age=30),
    User(id=2, name="Anna", age=25),
]

# GraphQL Query class
@strawberry.type
class Query:
    @strawberry.field
    def get_users(self) -> list[User]:
        return users

# GraphQL Mutation class (add new user)
@strawberry.type
class Mutation:
    @strawberry.mutation
    def create_user(self, name: str, age: int) -> User:
        new_user = User(id=len(users) + 1, name=name, age=age)
        users.append(new_user)
        return new_user

# Create GraphQL schema
schema = strawberry.Schema(query=Query, mutation=Mutation)

# Create FastAPI app
app = FastAPI()

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

```
uvicorn.exe main:app --reload
```

The GraphQL Playground will be available at the following URL: http://127.0.0.1:8000/graphql

---

Sample Task: Library System with GraphQL

Task Description

Create a simple GraphQL API to manage data in a library system. The system should store books and authors, and allow adding new books.

Requirements

The GraphQL schema should include the following types:

- **Author**:
  - `id` (Int)
  - `name` (String)
- **Book**:
  - `id` (Int)
  - `title` (String)
  - `author` (Author)
  - `year` (Int)

Features to Implement

- **Queries:**
  - List all books (title, author name, year)
  - List books by a specific author (searched by name)
- **Mutations:**
  - Add a new book with the following fields: title, author ID, year

Example Query

```
query {
  books {
    title
    author {
      name
    }
    year
  }
}
```

Example Mutation

```
mutation {
  addBook(title: "1984", authorId: 1, year: 1949) {
    id
    title
    author {
      name
    }
  }
}
```

Technical Requirements

- Use **FastAPI** and **Strawberry GraphQL** libraries
- Store data using built-in lists (e.g., `authors`, `books`)
- The GraphQL endpoint should be accessible at `http://127.0.0.1:8000/graphql`

Bonus

- Implement a new mutation to add an author by name
- Implement a query to list books by a given year