Why You Should Use **Implementation-First** to Build Your GraphQL Schema

Erik Wrede, fulfillmenttools

GraphQL Conf 2024

👋 I‘m Erik

- **Software Engineer** at **fulfillmenttools**
- **OSS Maintainer** GraphQL Python Strawberry & Graphene

Code-First or **Schema-First?**

Schema-First

Type Definitions

```
type Query {
  user(id: ID!): User
}

type User {
  id: ID
  profilePicture: Image
}
```

Schema-First

Type Definitions

```
type Query {
  user(id: ID!): User
}

type User {
  id: ID
  profilePicture: Image
}
```

Codegen

```
export type User = {
  id: string;
  profilePicture: Image;
};
```

Schema-First

Resolvers

```
@Resolver('User')
class UserResolver {

	@Query(() => 'User')
	async user(@Arg('id') id: number) : User {
		return userService.getUser(id);
	}

	@FieldResolver('profilePicture')
	async profilePicture(@Root() user: User) :	Image {
		return pictureService.getProfilePicture(user.id);
	}
}
```

Resolvers are separate from type definitions

Resolvers are separate from type definitions

Redundancy & Duplication

Code-First

```
interface User {
  id: number
  profilePicture: Image
}

const UserType = new GraphQLObjectType<User>({
        name: 'User',
        fields: {
                id: { type: GraphQLInt },
                profilePicture: {
                        type: ImageType
                        resolve: (user) => pictureService.getProfilePicture(user.id)
                }
        }
});
```

Code-First

```
interface User {
  id: number
  profilePicture: Image
}
```

duplication

```
const UserType = new GraphQLObjectType<User>({
        name: 'User',
        fields: {
                id: { type: GraphQLInt },
                profilePicture: {
                        type: ImageType
                        resolve: (user) => pictureService.getProfilePicture(user.id)
                }
        }
});
```

What about **consistency?**

Consistency

```
interface User {
  id: string;
  profilePicture: Image
}

const UserType = new GraphQLObjectType<User>({
  name: 'User‘,
  fields: {
    id: { type: GraphQLInt },
    profilePicture: {
      type: ImageType,
      args: {
        resolution: { type: GraphQLString }
      },
    resolve(user: User, {resaulution}) {
      return pictureService.getProfilePicture(user.id, resaulution) }
  }
});
```

Consistency

```
interface User {
  id: string;
  profilePicture: Image
}

const UserType = new GraphQLObjectType<User>({
  name: 'User‘,
  fields: {
    id: { type: GraphQLInt },
    profilePicture: {
      type: ImageType,
      args: {
        resolution: { type: GraphQLString }
      },
    resolve(user: User, {resaulution}) {
      return pictureService.getProfilePicture(user.id, resaulution) }
  }
});
```

Consistency

```
interface User {
  id: string;
  profilePicture: Image
}

const UserType = new GraphQLObjectType({
  name: 'User‘,
  fields: {
    id: { type: GraphQLInt },
    profilePicture: {
      type: ImageType,
      args: {
        resolution: { type: GraphQLString }
      },
    resolve(user: User, {resaulution}) {
      return pictureService.getProfilePicture(user.id, resaulution) }
        }
  }
});
```

Consistency II

Resolvers

```
@Resolver('User')
class UserResolver {

    @Query(() => 'User')
    async user(@Arg('id') id: number) : User {
        return userService.getUser(id);
    }

    @FieldResolver('profilePicture')
    async profilePicture(@Root() user: User) :    Image {
        return pictureService.getProfilePicture(user.id);
    }
}
```

This is not guaranteed to be of type User
profilePicture is on this type but not yet resolved!

This is not guaranteed to support the Schema Type for the field profilePicture

Conventional Frameworks **don’t ensure consistency** between schema types and resolvers

Conventional Frameworks **don’t ensure consistency** between schema types and resolvers

Type safety not guaranteed

Comparison

	Works without Codegen	No Duplication	Consistency of Resolver & Schema Type
Schema-First	✗ *	✗	✗
Code-First	✓	✗	✗

* Check out graphql.tada & GraphQLSP

GraphQL @ Facebook (2016)

```
<<GraphQLObjectType('User', 'A user of our app')>>
final class User {
        public function __construct(private string $id) {}

        <<GraphQLField('profilePicture', 'The profile picture')>>
        public function getProfilePicture(): ProfilePicture {
                return getProfilePicture($this->id);
        }
}
```

GraphQL @ Facebook (2016)

```
<<GraphQLObjectType('User', 'A user of our app')>>
final class User {
        public function __construct(private string $id) {}

        <<GraphQLField('profilePicture', 'The profile picture')>>
        public function getProfilePicture(): ProfilePicture {
                return getProfilePicture($this->id);
        }
}
```

Single source of truth for
type definition & resolvers

GraphQL @ Facebook (2016)

```
<<GraphQLObjectType('User', 'A user of our app')>>
final class User {
        public function __construct(private string $id) {}

        <<GraphQLField('profilePicture', 'The profile picture')>>
        public function getProfilePicture(): ProfilePicture {
                return getProfilePicture($this->id);
        }
}
```

Single source of truth for type definition & resolvers

No **duplication** or **inconsistency**

GraphQL @ Facebook (2016)

```
<<GraphQLObjectType('User', 'A user of our app')>>
final class User {
	public function __construct(private string $id) {}

	<<GraphQLField('profilePicture', 'The profile picture')>>
	public function getProfilePicture(): ProfilePicture {
		return getProfilePicture($this->id);
	}
}
```

Schema type is inferred
from method return type

GraphQL @ Facebook (2016)

```
<<GraphQLObjectType('User', 'A user of our app')>>
final class User {
	public function __construct(private string $id) {}

	<<GraphQLField('profilePicture', 'The profile picture')>>
	public function getProfilePicture(): ProfilePicture {
		return getProfilePicture($this->id);
	}
}
```

Schema type is inferred
from method return type

Type safe by design

Implementation-First GraphQL

Implementation-First GraphQL

- Derives the schema from the **implementation**
- Generates GraphQL types from the native types, fields & **methods**
- Leverages **annotations** or **reflection** to define the schema
- Infers an **isomorphic GraphQL schema**
- Is **type safe** by design

Implementation-First Frameworks

Strawberry

HotChocolate

Grats

Implementation-First GraphQL

```
@strawberry.type
class User:
  id: str

  @strawberry.field
  def profile_picture(self) -> Image:
    ...
```

Types in Implementation-First

```
@strawberry.type
class User:
  id: str

  @strawberry.field
  def profile_picture(self, resolution: str) -> Image:
    ...
```

Types in Implementation-First

```
@strawberry.type
class User:
  id: str

  @strawberry.field
  def profile_picture(self, resolution: str) -> Image:
    ...
```

This is guaranteed to be a User Object

Types in Implementation-First

```
@strawberry.type
class User:
  id: str

  @strawberry.field
  def profile_picture(self, resolution: str) -> Image:
    ...

  @strawberry.field
  def friends(self) -> list[User | None]:
    ...
```

Types in Implementation-First

```
@strawberry.type
class User:
  id: str

  @strawberry.field
  def profile_picture(self, resolution: str) -> Image:
    ...

  @strawberry.field
  def friends(self) -> list[User | None]:
    ...
```

GraphQL Schema type is inferred from field type

Schema type is inferred from method return type

Ignoring Fields

```
@strawberry.type
class User:
  id: str
  password_hash: strawberry.Private[str]

  @strawberry.field
  def profile_picture(self) -> Image:
    ...
```

Scalars

… correspond to primitives or types in your language

```
BigInt = strawberry.scalar(
  Union[int, str],
  serialize=lambda v: int(v),
  parse_value=lambda v: str(v),
  description="BigInt field",
)

@strawberry.type
class Post:
  view_count: BigInt
```

Queries & Mutations

… are also fields on object types

```
@strawberry.type
class Query:

  @strawberry.field
  def user(self, id: strawberry.ID) -> User:
                return User(id=id, name="Erik")


@strawberry.type
class Mutation:

  @strawberry.mutation
  def createUser(input: CreateUserInput) -> CreateUserResponse:
    ...
```

Will Implementation-First make my schema harder to understand across languages?

No.

Strawberry

```
@strawberry.type
class User:
  id: str
  password_hash: strawberry.Private[str]

  @strawberry.field
  def profile_picture(self) -> Image:
    ...
```

Hot Chocolate (Annotation-Based)

```
public class User
{
  public string Id { get; set; }

  [GraphQLIgnore]
  public string passwordHash { get; set; }

  public Image GetProfilePicture()
  {
    ...
  }
}
```

Grats

```
/**
* A user in our system
* @gqlType
*/
class User {

  /** @gqlField */
  id: string;

  passwordHash: string;

  /** @gqlField */
  profilePicture(): Image {
    ...
  }
}
```

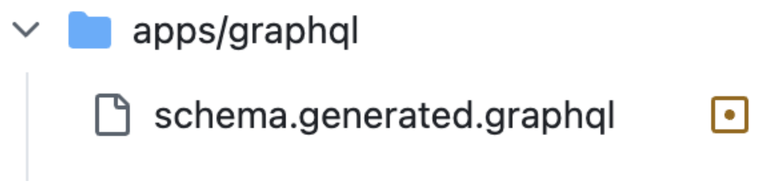
Thinking **Implementation-First**

Thinking **Implementation-First**

Your **Data Objects** already form a **Graph**.
With **Implementation-First**, you simply choose the **fields** and **methods** to expose.

Still unclear for new devs?

Just Check in your SDL


apps/graphql

schema.generated.graphql

Advanced Patterns

Native Generics

```
T = TypeVar("T")

@strawberry.type
class Connection(Generic[T]):
  edges: list[Edge[T]]

  @strawberry.field
  def pageInfo(self) -> PageInfo:
    return PageInfo(endCursor=self.edges[-1].cursor)


@strawberry.type
class Query:
  users: Connection[User]
```

Native Generics

```
type UserConnection {
  pageInfo: PageInfo
  edges: [UserEdge]
}

type UserEdge {
  cursor: String
  node: User
}
```

Field Extensions

```
input CreateUserInput {
  name: String!
  email: String!
}

type Mutation {
  createUser(input: CreateUserInput!): CreateUserResponse!
}
```

Field Extensions

```
@straberry.input
class CreateUserInput:
  name: str
  email: str

@strawberry.type
class Mutation:
  def createUser(input: CreateUserInput) -> CreateUserResponse:
    ...
```

Field Extensions

```
@strawberry.type
class Mutation:
  @strawberry.mutation(extensions=[InputMutationExtension()])
  def createUser(self, name: str, email: str) -> CreateUserResponse:
    ...
```

Comparison

	Works without Codegen	No Duplication	Consistency of Resolver & Schema Type
Schema-First	✗	✗	✗
Code-First	✓	✗	✗
Implementation-First	✓	✓	✓

Why should you use Implementation-First?

- **Intuitive** & Fast approach to building GraphQL APIs
- Leverages native language features
- **Removes duplication** and ensures **consistency**
- **Production-Ready**
- **Type-Safe** by design

Credits

- Jordan Eldredge (Grats, Meta)
- Michael Staib (ChiliCream)
- Adam D.I. Kramer’s Talk
  on GraphQL @ Facebook

Thank you!

Check out the docs

http://strawberry.rocks

http://grats.capt.dev

You can find me on

erikwrede

erikwrede