GraphQL and Sangria

How to get a GraphQL API Server Up and Running

Daniel Brice, CJ Affiliate (dbrice@cj.com) David Ron, Pay Junction

5 March 2018

Designing Your GraphQL Schema

Designing Your GraphQL Schema (1)

Database Layout

```
TABLE foo
    INT    id,
    STRING name,
    INT    bar.id

TABLE bar
    INT    id,
    STRING name
```

GraphQL Schema

```
type Foo {
    id:    Int!
    name:  String!
    barId: Int!
}
type Bar {
    id:   Int!
    name: String!
}
type Query {
    foos: [Foo!]!
    bars: [Bar!]!
}
```

Designing Your GraphQL Schema (2)

Database Layout

```
TABLE foo
    INT    id,
    STRING name,
    INT    bar.id

TABLE bar
    INT    id,
    STRING name
```

GraphQL Schema

```
type Foo {
    id:   Int!
    name: String!
    bar:  Bar!
}
type Bar {
    id:   Int!
    name: String!
}
type Query {
    foos: [Foo!]!
    bars: [Bar!]!
}
```

Designing Your GraphQL Schema (3)

Database Layout

```
TABLE foo
    INT    id,
    STRING name,
    INT    bar.id

TABLE bar
    INT    id,
    STRING name
```

GraphQL Schema

```
type Foo {
    id:   Int!
    name: String!
    bar:  Bar!
}
type Bar {
    id:   Int!
    name: String!
    foos: [Foo!]!
}
type Query {
    foos: [Foo!]!
    bars: [Bar!]!
}
```

Designing Your GraphQL Schema (4)

Database Layout

```
TABLE foo
    INT    id,
    STRING name,
    INT    bar.id

TABLE bar
    INT    id,
    STRING name
```

GraphQL Schema

```
type Foo {
    id:   Int!
    name: String!
    bar:  Bar!
}
type Bar {
    id:   Int!
    name: String!
    foos: [Foo!]!
}
type Query {
    foos(ids: [Int!]): [Foo!]!
    bars(ids: [Int!]): [Bar!]!
}
```

Executing GraphQL Queries

Executing GraphQL Queries

```
// Global constant.
val schema: Schema[DAO, Unit] = ...

// Create in response to incoming request.
val : DAO = ...

// Contained in POST body of incoming request.
val unparsedQuery: String = ...

// May contain a SyntaxError
val parsedQuery: Try[Document] = QueryParser.parse(unparsedQuery)

// May contain a ValidationError
val futureResult: Future[Json] = Executor.execute(
  queryAst    = parsedQuery.get,
  userContext = dao,
  schema      = schema
)
```

Defining Your Data Layer

Defining Your Data Layer

Database Layout

```
TABLE foo
    INT    id,
    STRING name,
    INT    bar.id

TABLE bar
    INT    id,
    STRING name
```

Data Layer

```
case class Foo( id:    Int,
                name:  String,
                barId: Int     )

case class Bar( id:   Int,
                name: String )

trait DAO {}
```

Implementing Your Schema

Implementing Your Schema (1)

Data Layer

```
case class Foo( id:    Int,
                name:  String,
                barId: Int     )

trait DAO {}
```

GraphQL Schema

```
type Foo {
    id:   Int!
    name: String!
    bar:  Bar!
}
```

Sangria Schema Implementation

```
lazy val foo: GqlObject[DAO, Foo] = ???
```

Implementing Your Schema (2)

Data Layer

```
case class Foo( id:    Int,
                name:  String,
                barId: Int     )

trait DAO {}
```

GraphQL Schema

```
type Foo {
    id:   Int!
    name: String!
    bar:  Bar!
}
```

Sangria Schema Implementation

```
lazy val foo: GqlObject[DAO, Foo] =
  deriveObjectType[DAO, Foo]()
```

Implementing Your Schema (3)

Data Layer

```
case class Foo( id:    Int,
                name:  String,
                barId: Int     )

trait DAO {}
```

GraphQL Schema

```
type Foo {
    id:   Int!
    name: String!
    bar:  Bar!
}
```

Sangria Schema Implementation

```
lazy val foo: GqlObject[DAO, Foo] =
  deriveObjectType[DAO, Foo](
    ReplaceField(
      fieldName = "barId",
      field     = ???
    )
  )
```

Implementing Your Schema (4)

Data Layer

```
case class Foo( id:    Int,
                name:  String,
                barId: Int     )

trait DAO {}
```

GraphQL Schema

```
type Foo {
    id:   Int!
    name: String!
    bar:  Bar!
}
```

Sangria Schema Implementation

```
lazy val foo: GqlObject[DAO, Foo] =
  deriveObjectType[DAO, Foo](
    ReplaceField(
      fieldName = "barId",
      field     = GqlField(
        name      = "bar",
        fieldType = bar,
        resolve   = cc => ???
      )
    )
  )
```

Implementing Your Schema (5)

Data Layer

```
case class Foo( id:    Int,
                name:  String,
                barId: Int     )

trait DAO {
  def fooBar(foo: Foo): Bar
}
```

GraphQL Schema

```
type Foo {
    id:   Int!
    name: String!
    bar:  Bar!
}
```

Sangria Schema Implementation

```
lazy val foo: GqlObject[DAO, Foo] =
  deriveObjectType[DAO, Foo](
    ReplaceField(
      fieldName = "barId",
      field     = GqlField(
        name      = "bar",
        fieldType = bar,
        resolve   = cc =>
          cc.ctx.fooBar(cc.value)
      )
    )
  )
```

Implementing Your Schema (6)

Data Layer

```
case class Bar( id:   Int,
                name: String )

trait DAO {
  def fooBar(foo: Foo): Bar
  def barFoos(bar: Bar): Foo
}
```

GraphQL Schema

```
type Bar {
    id:   Int!
    name: String!
    foos: [Foo!]!
}
```

Sangria Schema Implementation

```
lazy val bar: GqlObject[DAO, Bar] =
  deriveObjectType[DAO, Bar](
    AddFields(
      GqlField(
        name =      "foos",
        fieldType = GqlList(foo),
        resolve =   cc =>
          cc.ctx.barFoos(cc.value)
      )
    )
  )
```

Implementing Your Schema (7)

Data Layer

```
trait DAO {
  ...
  def queryFoos()
  def queryBars()
}
```

GraphQL Schema

```
type Query {
  foos(ids: [Int!]): [Foo!]!
  bars(ids: [Int!]): [Bar!]!
}
```

Sangria Schema Implementation

```
lazy val query: GqlObject[DAO, Unit] = ???
```

Implementing Your Schema (8)

Data Layer

```
trait DAO {
  ...
  def queryFoos(
    ids: Option[Seq[Int]]): Seq[Foo]

  def queryBars(
    ids: Option[Seq[Int]]): Seq[Bar]
}
```

GraphQL Schema

```
type Query {
  foos(ids: [Int!]): [Foo!]!
  bars(ids: [Int!]): [Bar!]!
}
```

Sangria Schema Implementation

```
lazy val query: GqlObject[DAO, Unit] = ???
```

Implementing Your Schema (9)

Data Layer

```
trait DAO {
  ...
  def queryFoos(
    ids: Option[Seq[Int]]): Seq[Foo]

  def queryBars(
    ids: Option[Seq[Int]]): Seq[Bar]
}
```

GraphQL Schema

```
type Query {
  foos(ids: [Int!]): [Foo!]!
  bars(ids: [Int!]): [Bar!]!
}
```

Sangria Schema Implementation

```
lazy val ids:
  GqlArgument[Option[Seq[Int]]] =
    GqlArgument(
      name         = "ids",
      argumentType =
        GqlOptionInput(GqlListInput(GqlInt
    )

lazy val query: GqlObject[DAO, Unit] = ???
```

Implementing Your Schema (10)

Data Layer

```
trait DAO {
  ...
  def queryFoos(
    ids: Option[Seq[Int]]): Seq[Foo]

  def queryBars(
    ids: Option[Seq[Int]]): Seq[Bar]
}
```

GraphQL Schema

```
type Query {
  foos(ids: [Int!]): [Foo!]!
  bars(ids: [Int!]): [Bar!]!
}
```

Sangria Schema Implementation

```
lazy val ids = ...

lazy val query: GqlObject[DAO, Unit] =
  GqlObject(
    name   = "Query",
    fields = gqlFields[DAO, Unit](
      ???, // foos field
      ???  // bars field
    )
  )
```

Implementing Your Schema (11)

Data Layer

```
trait DAO {
  ...
  def queryFoos(
    ids: Option[Seq[Int]]): Seq[Foo]

  def queryBars(
    ids: Option[Seq[Int]]): Seq[Bar]
}
```

GraphQL Schema

```
type Query {
  foos(ids: [Int!]): [Foo!]!
  bars(ids: [Int!]): [Bar!]!
}
```

Sangria Schema Implementation

```
lazy val query: GqlObject[DAO, Unit] =
  GqlObject(
    name   = "Query",
    fields = gqlFields[DAO, Unit](
      GqlField(
        name      = "foos",
        fieldType = GqlList(foo),
        arguments = List(ids),
        resolve   = cc => ???
      ),
      GqlField(
        name      = "bars",
        fieldType = GqlList(bar),
        arguments = List(ids),
        resolve   = cc => ???
      )
```

Implementing Your Schema (12)

Data Layer

```
trait DAO {
  ...
  def queryFoos(
    ids: Option[Seq[Int]]): Seq[Foo]

  def queryBars(
    ids: Option[Seq[Int]]): Seq[Bar]
}
```

GraphQL Schema

```
type Query {
  foos(ids: [Int!]): [Foo!]!
  bars(ids: [Int!]): [Bar!]!
}
```

Sangria Schema Implementation

```
lazy val query: GqlObject[DAO, Unit] =
  GqlObject(
    name   = "Query",
    fields = gqlFields[DAO, Unit](
      GqlField(
        ...
        resolve   = cc =>
          cc.ctx.queryFoos(cc.arg(ids))
      ),
      GqlField(
        ...
        resolve   = cc =>
          cc.ctx.queryBars(cc.arg(ids))
      )
    )
  )
```