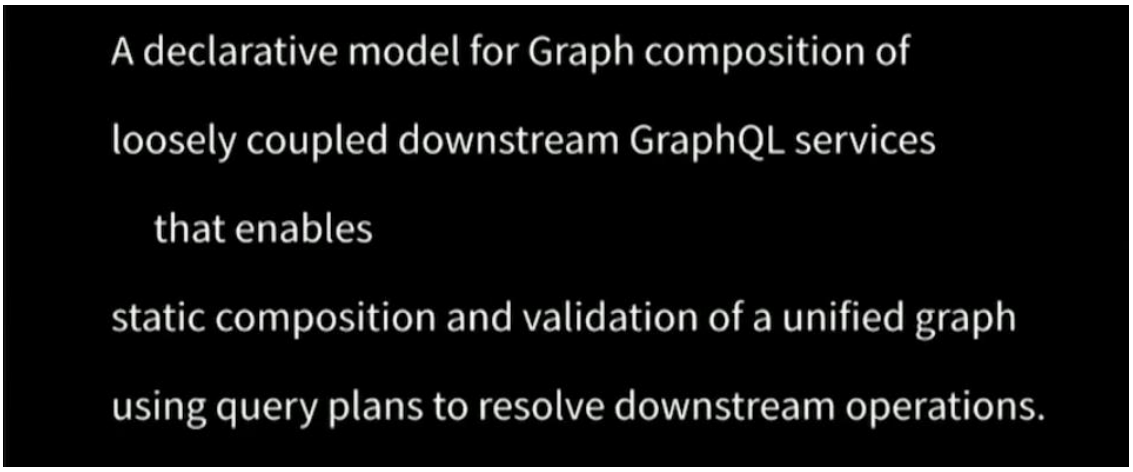
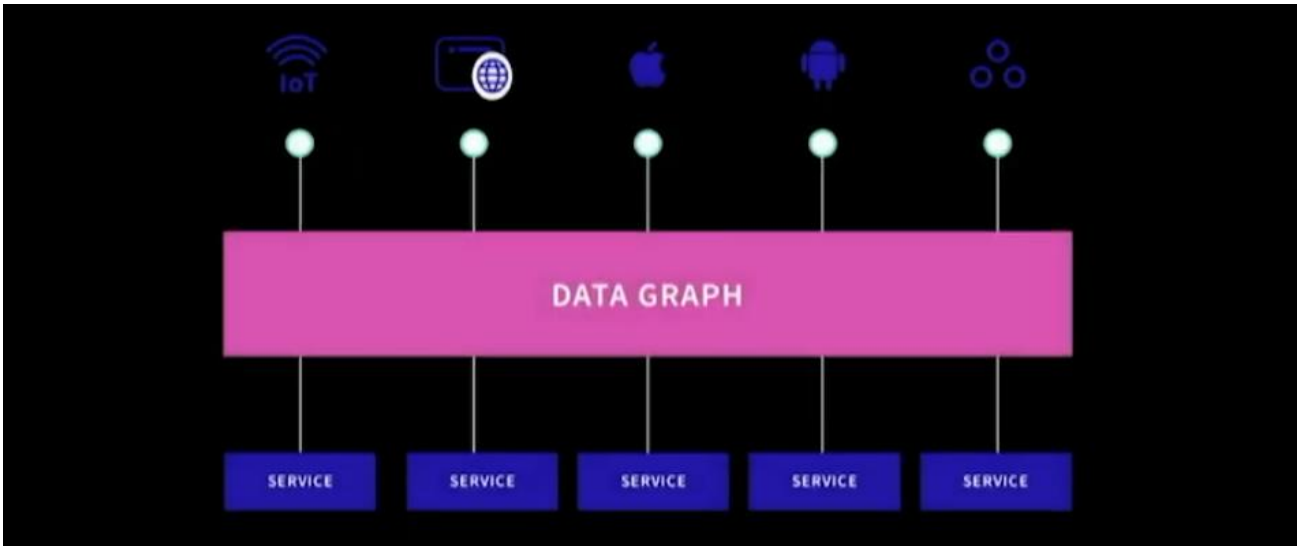
The Architecture of Federation

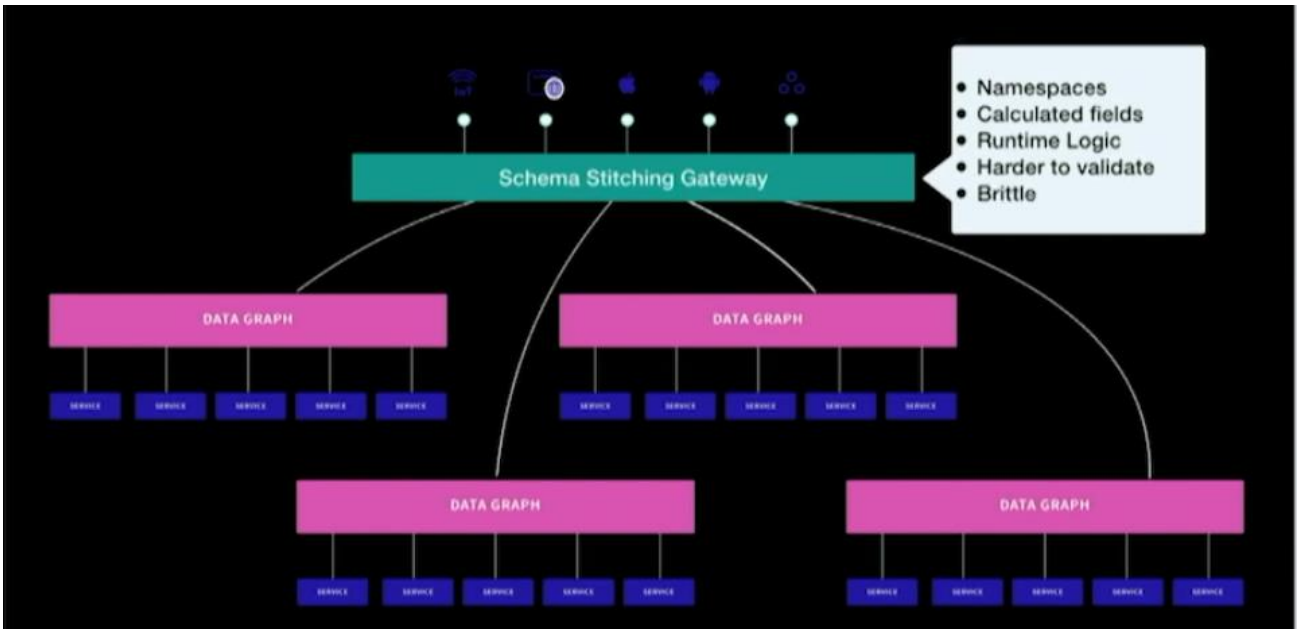
Apollo Federation is a new architecture for GraphQL, where you can now divide the implementation of a data graph across distinct, composable services. With this approach, different teams can collaborate across a single graph whose implementation is loosely-coupled. Jeff will cover the architectural principles behind federation and show how you can implement a well-designed and structured data graph using Apollo.



Part 1 - Motivation

Framing the problem





First Principles

Principled GraphQL

"Though there is only one graph, the implementation of that graph should be federated across multiple teams."

From PrincipledGraphQL.com, Matt DeBergalis and Geoff Schmidt

Is this for me?

HYPE, YAGNI, etc

More than one graph? Modularity?
Different security, performance, CI/CD requirements?
Adding more teams?

- Adoption is incremental
- Implementation is simple
- Good language support 
- Clear path to scale

Part 2 - Design

How does it work?

Federation is Different

Declarative
Separates Concerns
“Just GraphQL”

A declarative model for Graph composition of

loosely coupled downstream GraphQL services

that enables

static composition and validation of a unified graph

using query plans to resolve downstream operations.

Declarative

GraphQL Syntax, Valid SDL
Static composition & validation
“Known Good” Schema Artifact
Benefit from ahead-of-time processing

A declarative model for Graph composition of

loosely coupled downstream GraphQL services

that enables

static composition and validation of a unified graph

using query plans to resolve downstream operations.

Separates Concerns

Lets Teams Prioritize
Logic in Services
Align with organization
Make stronger technical choices

A declarative model for Graph composition of

loosely coupled downstream GraphQL services

that enables

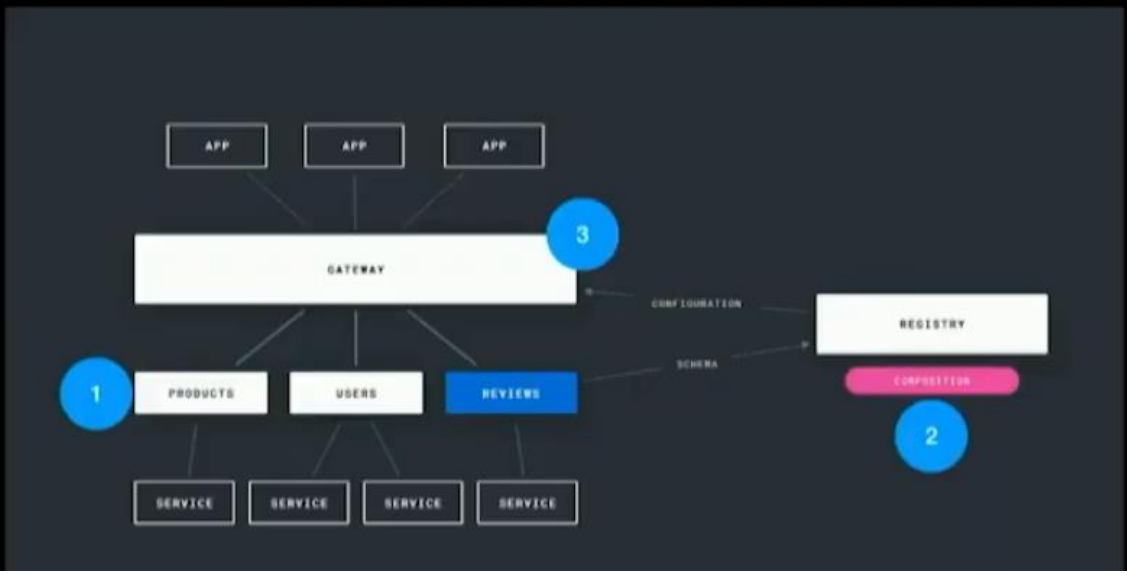
static composition and validation of a unified graph

using query plans to resolve downstream operations.

“Just GraphQL”

Uses SDL primitive
Opaque to consumers
Tooling “just works”
No changes to clients

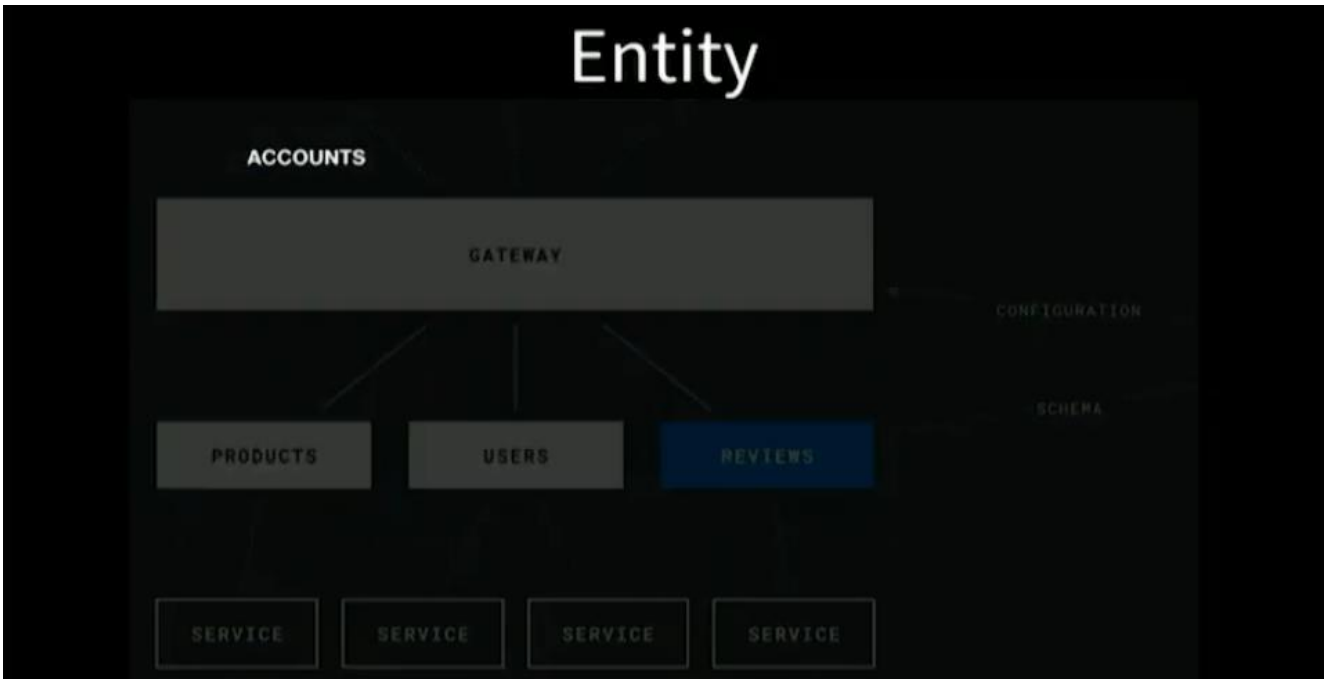
Basics of Federation

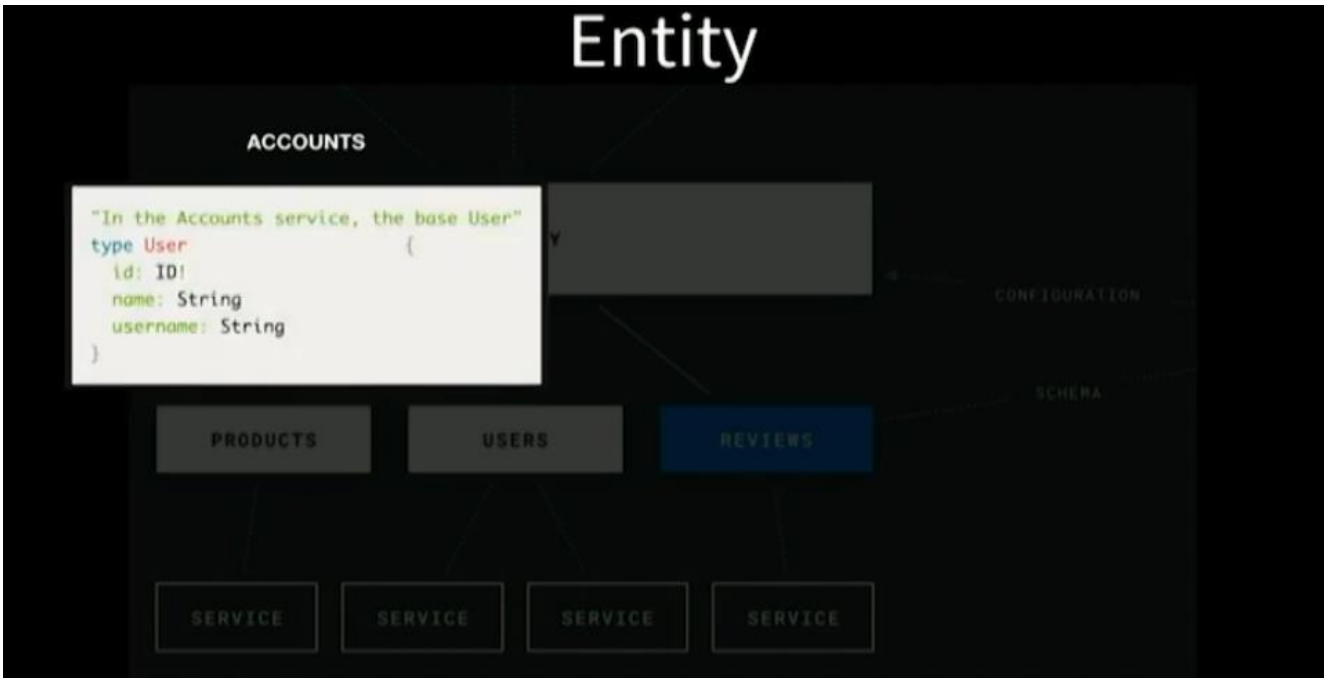


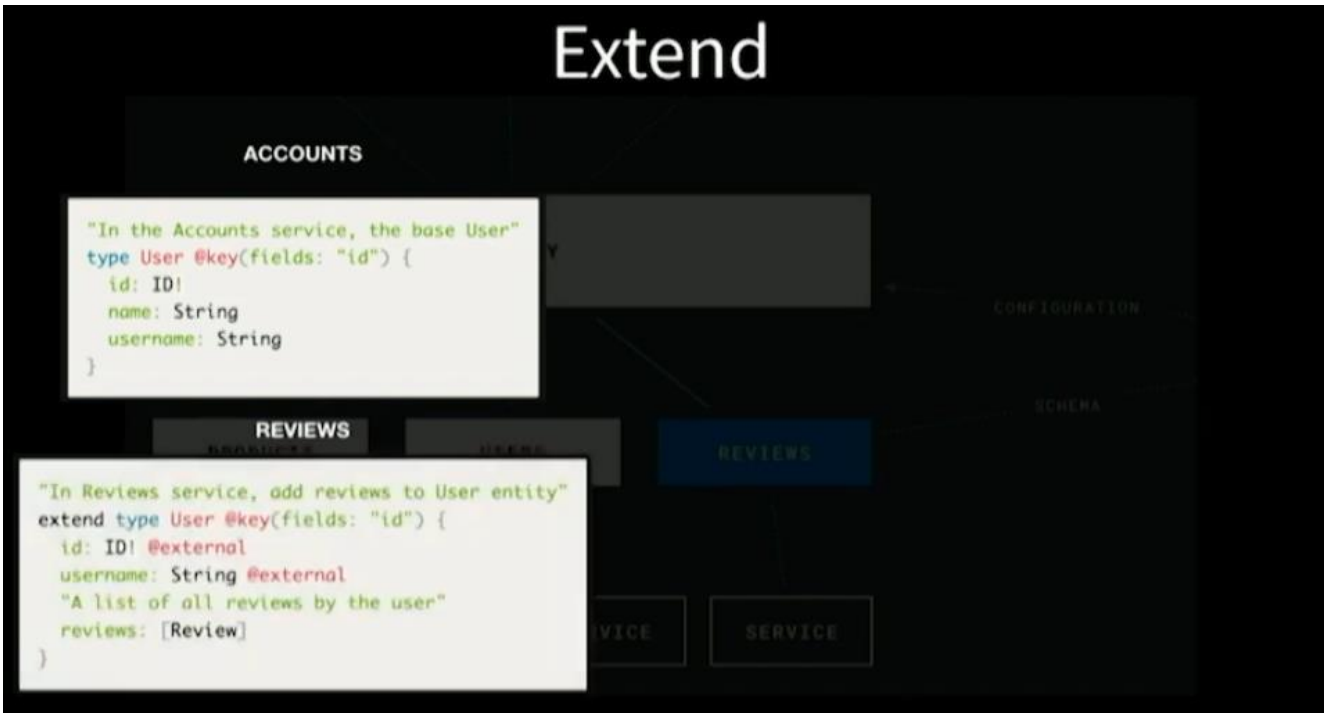
Part 3 - Implementation

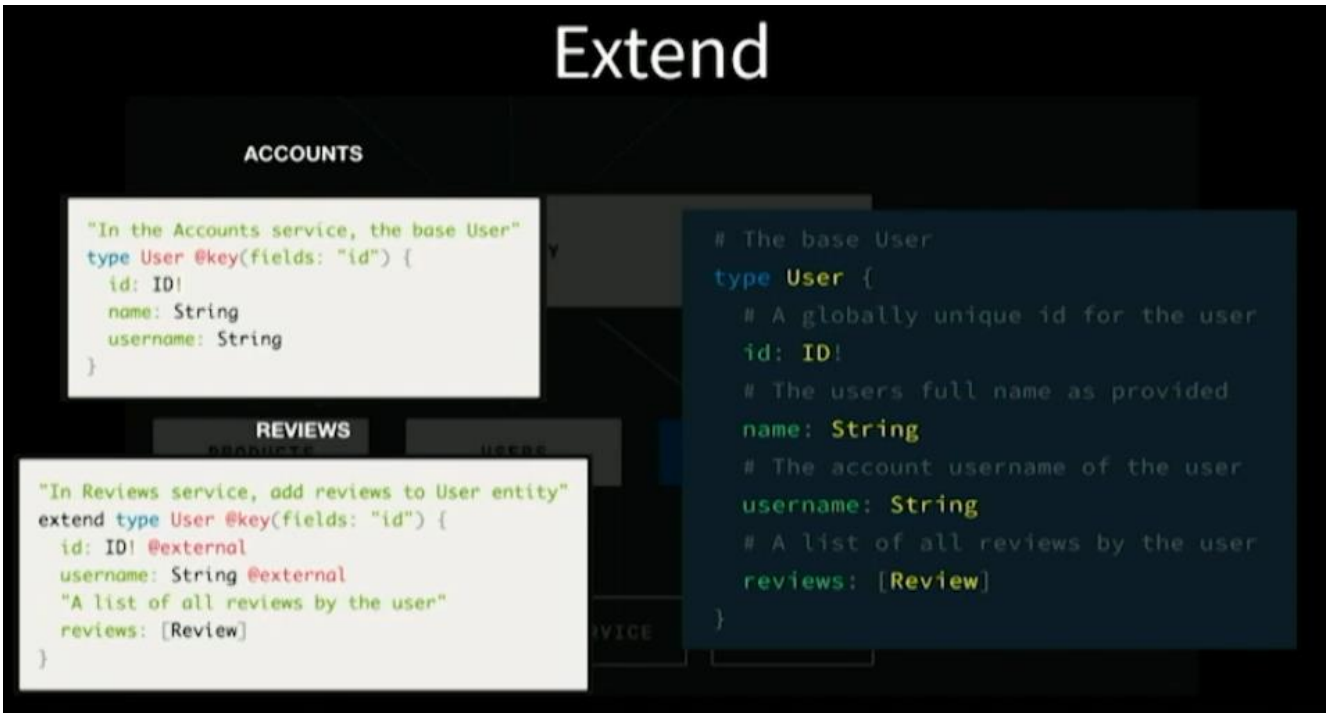
Enough with the talk, show me how it works

- **Reference** - find a type in another service
- **Extend** - add richness to those types
- **Query Plan** - take a single operation and call each service









When you view this schema in playground, it is a single unified graph federated in implementation

A declarative model for Graph composition of

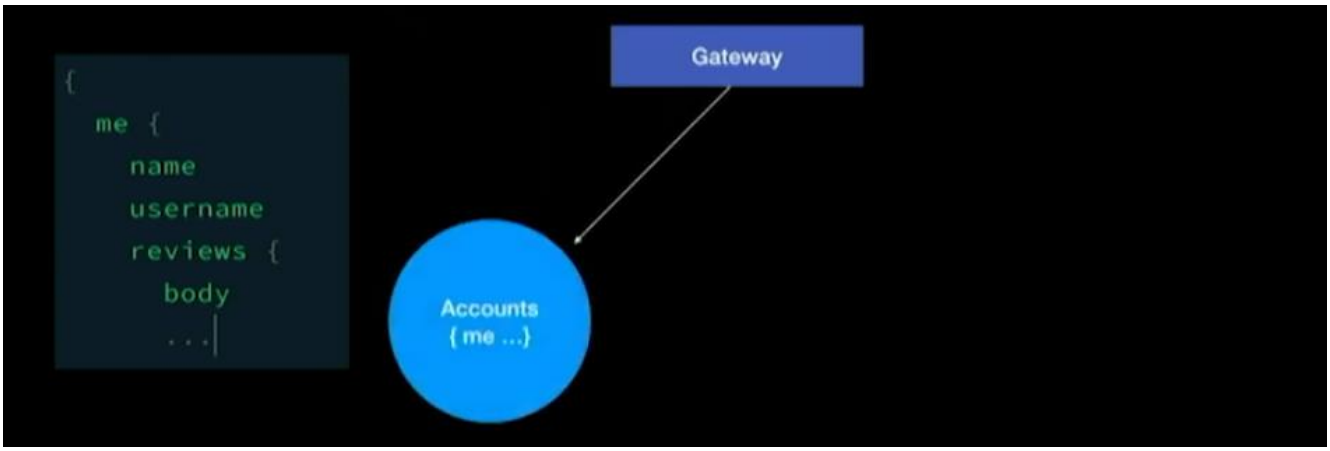
loosely coupled downstream GraphQL services

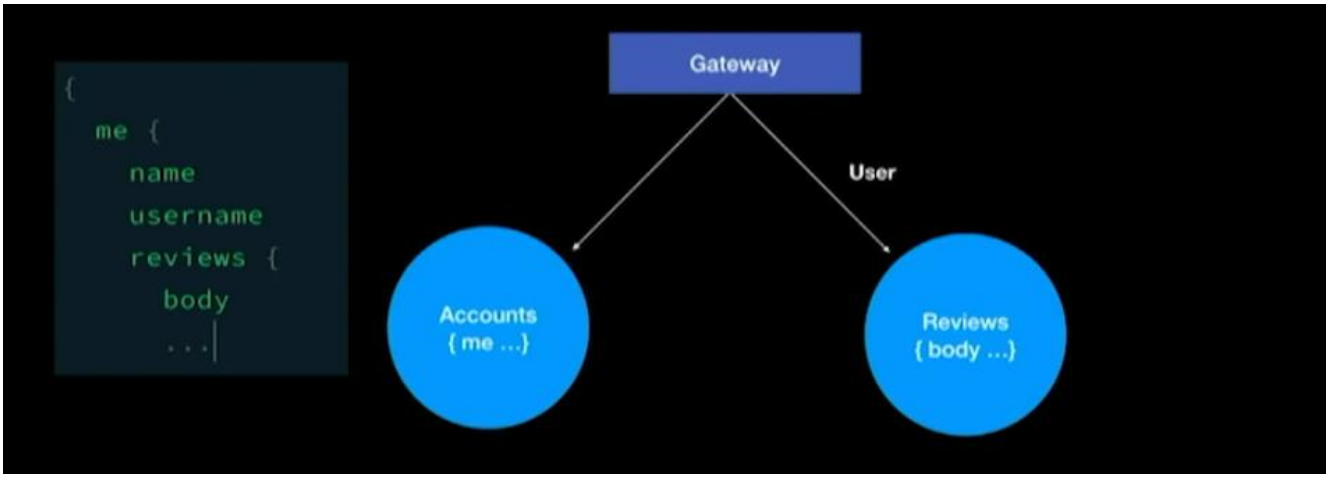
that enables

static composition and validation of a unified graph

using query plans to resolve downstream operations.

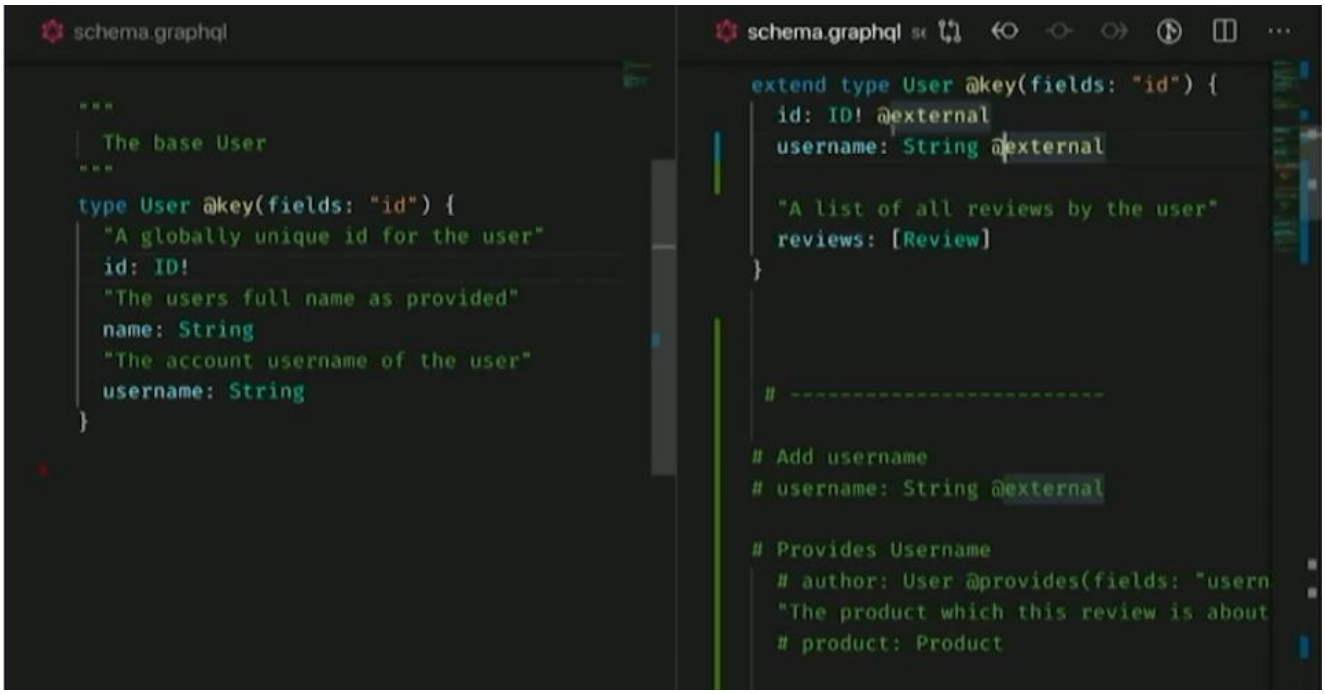
```
{
  me {
    name
    username
    reviews {
      body
      ...
```

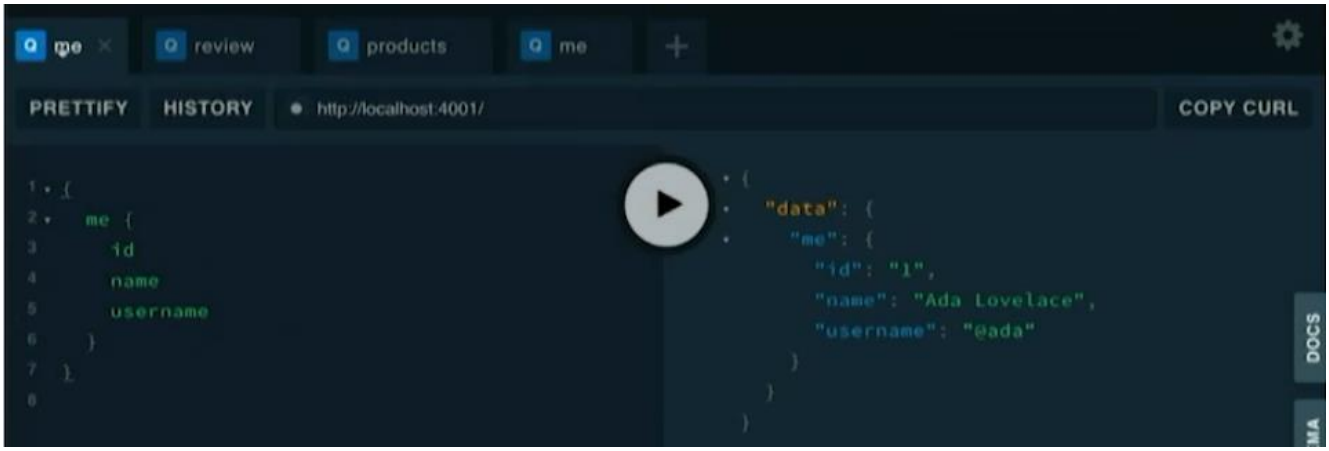


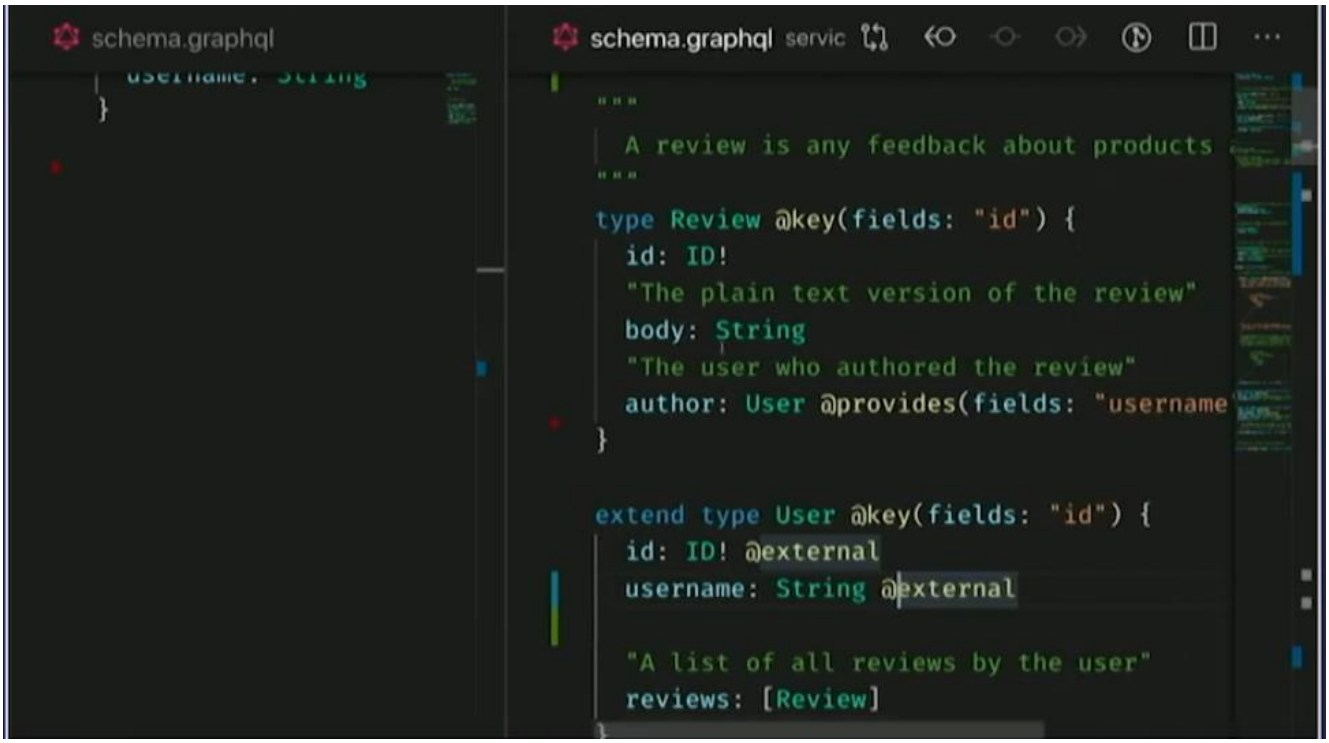


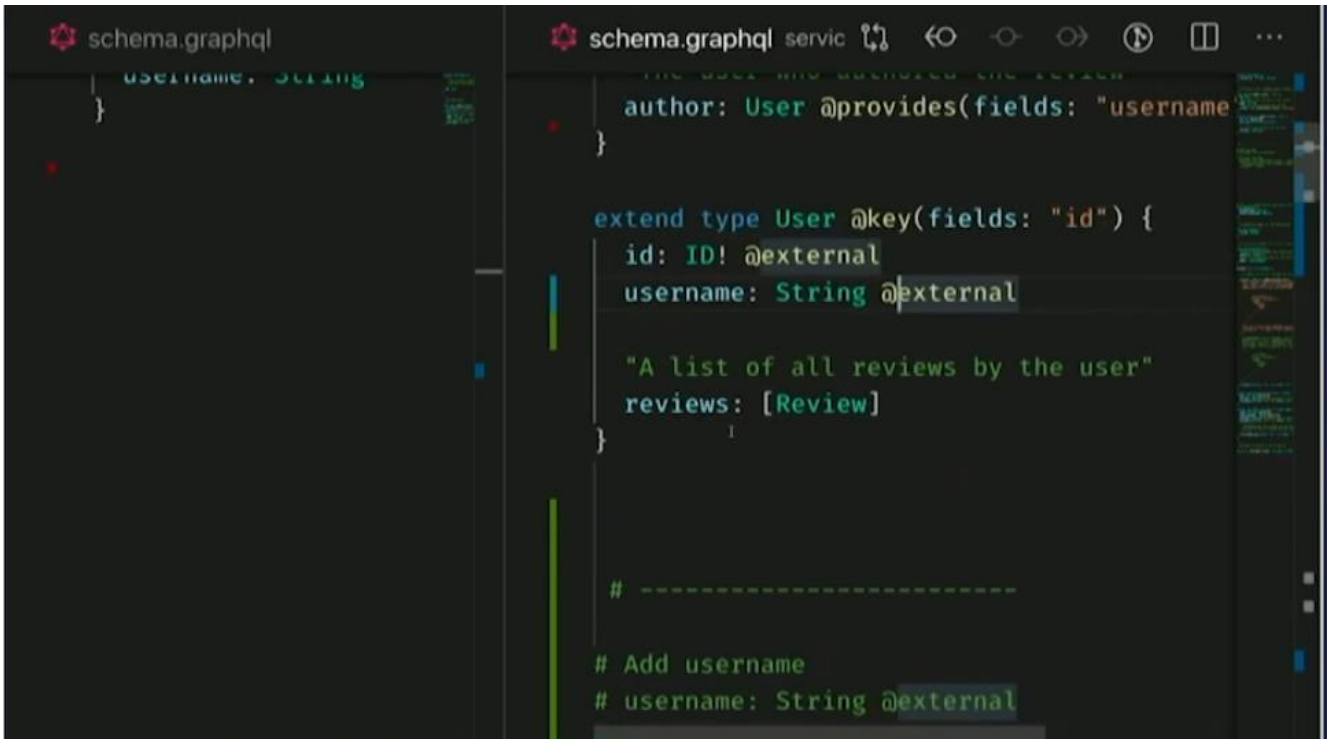
DEMO

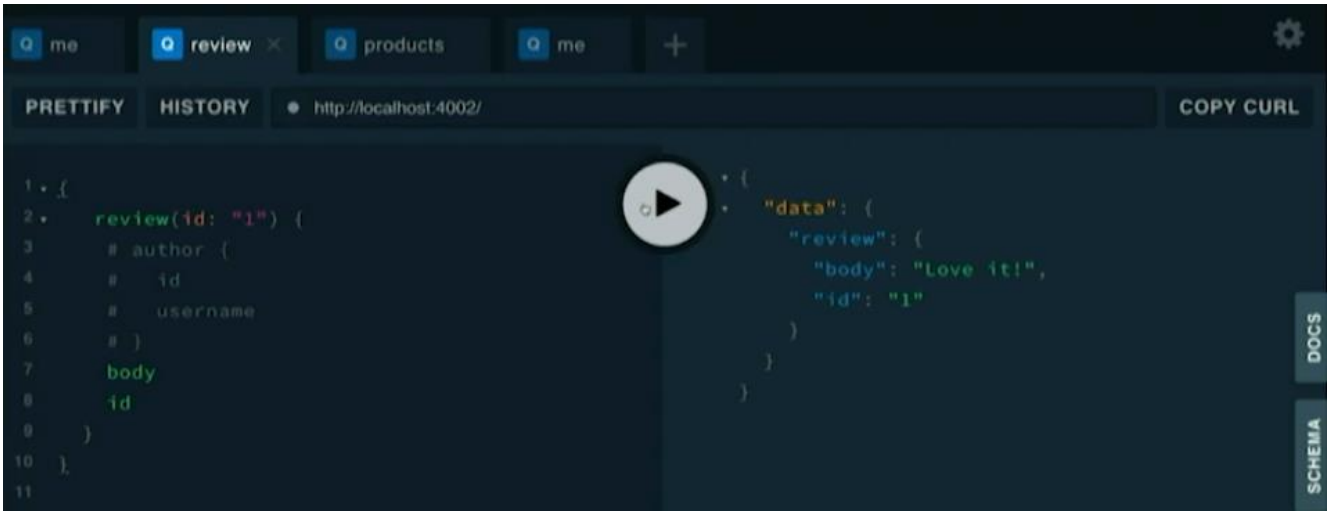
Entities
Extension
Query Plan
Magic

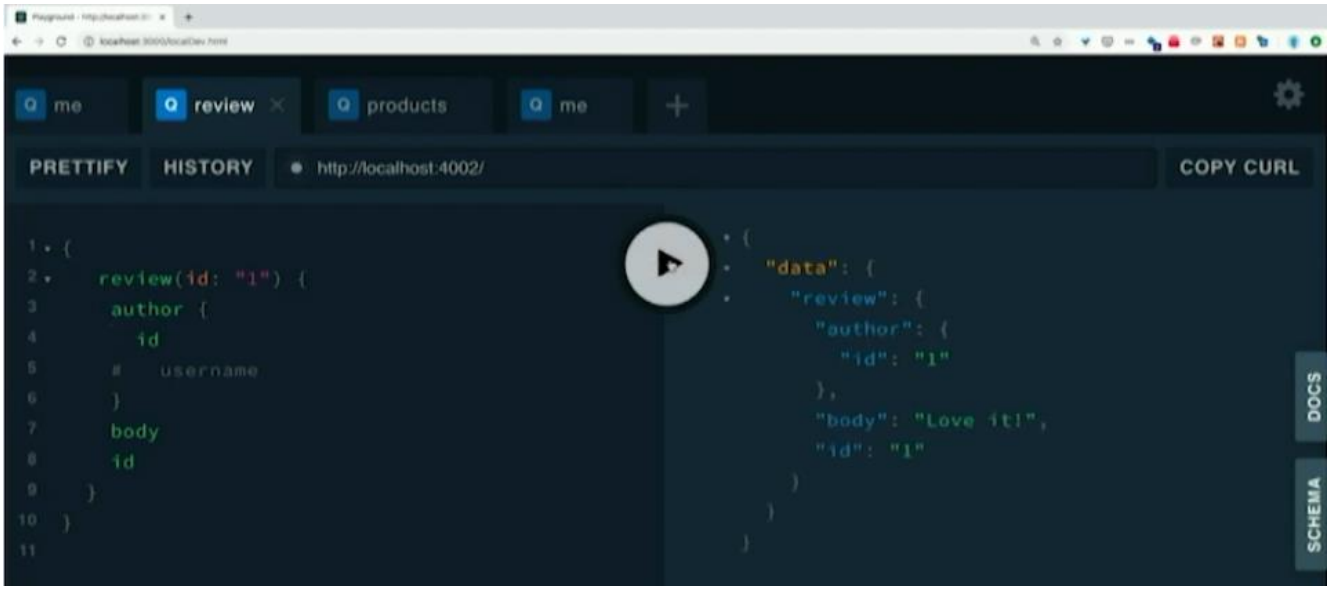

schema.graphql
"""
The base User
"""
type User @key(fields: "id") {
"A globally unique id for the user"
id: ID!
"The users full name as provided"
name: String
"The account username of the user"
username: String
}
schema.graphql
extend type User @key(fields: "id") {
id: ID! @external
username: String @external
"A list of all reviews by the user"
reviews: [Review]
}
# Add username
# username: String @external
# Provides Username
# author: User @provides(fields: "usern
"The product which this review is about
product: Product

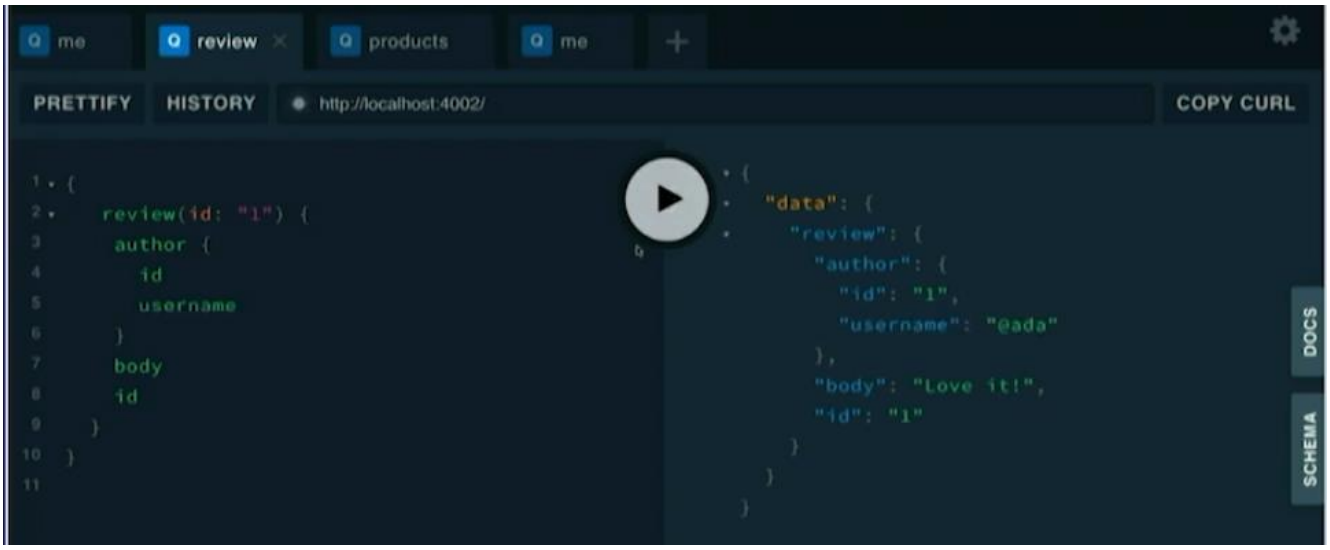

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4001/
COPY CURL
{
me {
id
name
username
}
}
{
"data": {
"me": {
"id": "1",
"name": "Ada Lovelace",
"username": "@ada"
}
}
}
DOCS

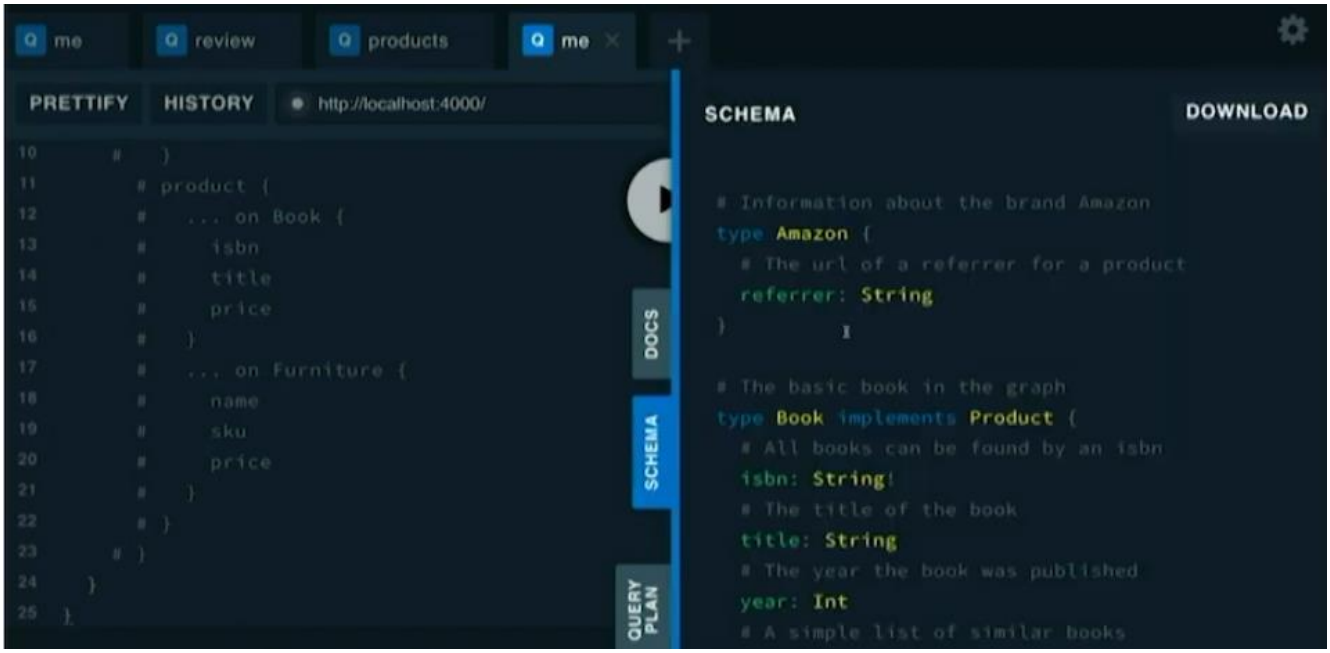

schema.graphql
}
schema.graphql servic
"""
A review is any feedback about products
"""
type Review @key(fields: "id") {
id: ID!
"The plain text version of the review"
body: String
"The user who authored the review"
author: User @provides(fields: "username
}
extend type User @key(fields: "id") {
id: ID! @external
username: String @external
"A list of all reviews by the user"
reviews: [Review]

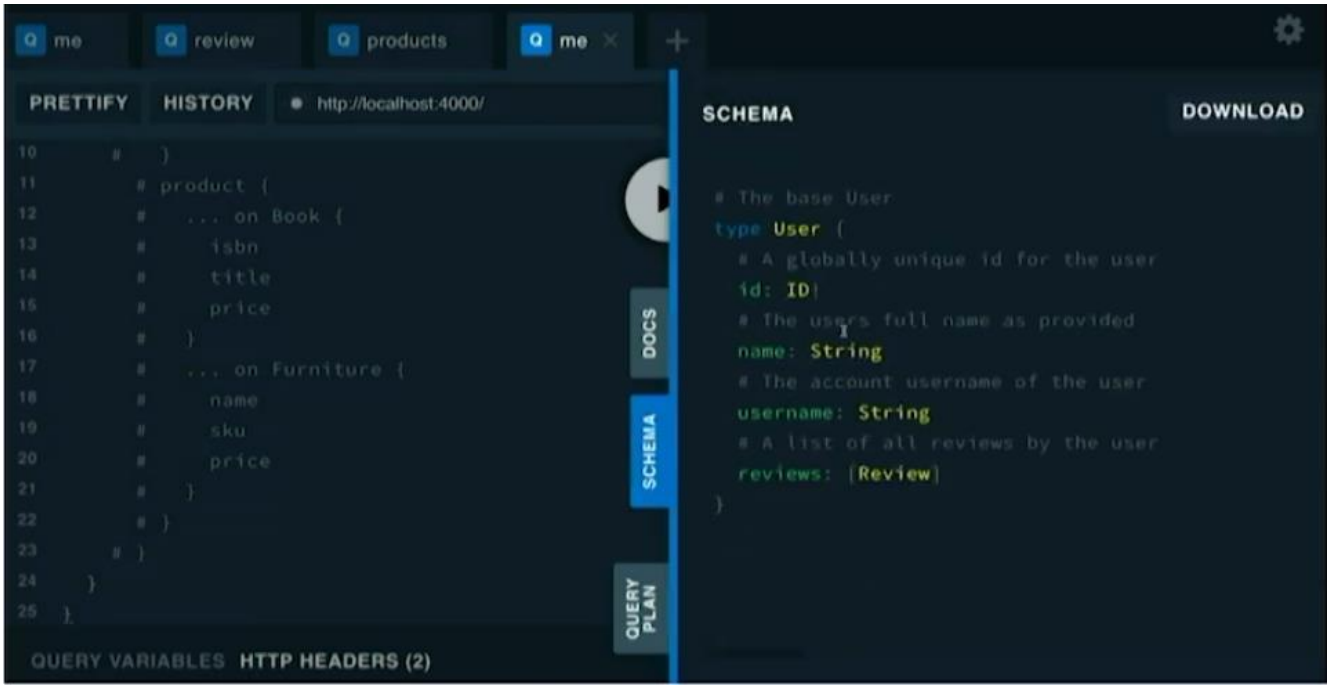

schema.graphql
schema.graphql servic
author: User @provides(fields: "username
}
extend type User @key(fields: "id") {
id: ID! @external
username: String @external
"A list of all reviews by the user"
reviews: [Review]
}
# ------------------------
# Add username
# username: String @external

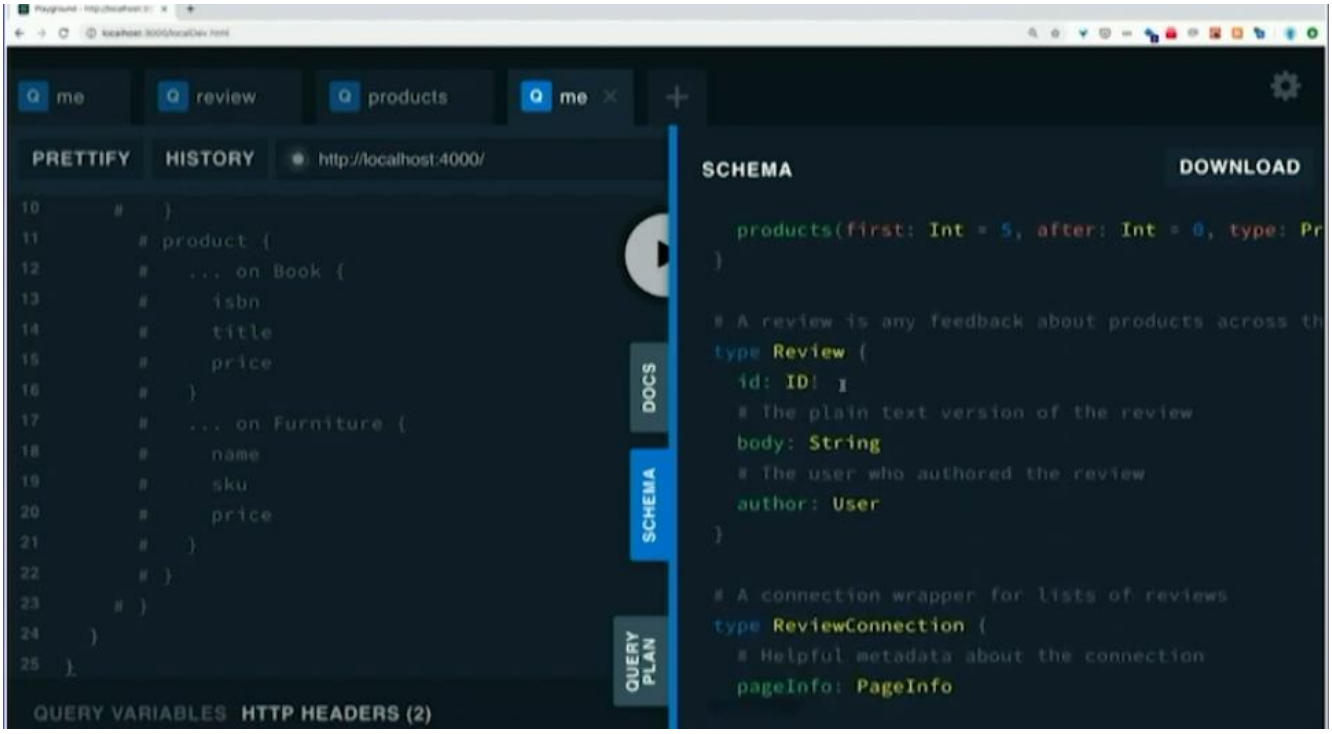

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4002/
COPY CURL
{
review(id: "1") {
# author {
# id
# username
# }
body
id
}
}
{
"data": {
"review": {
"body": "Love it!",
"id": "1"
}
}
}
DOCS
SCHEMA

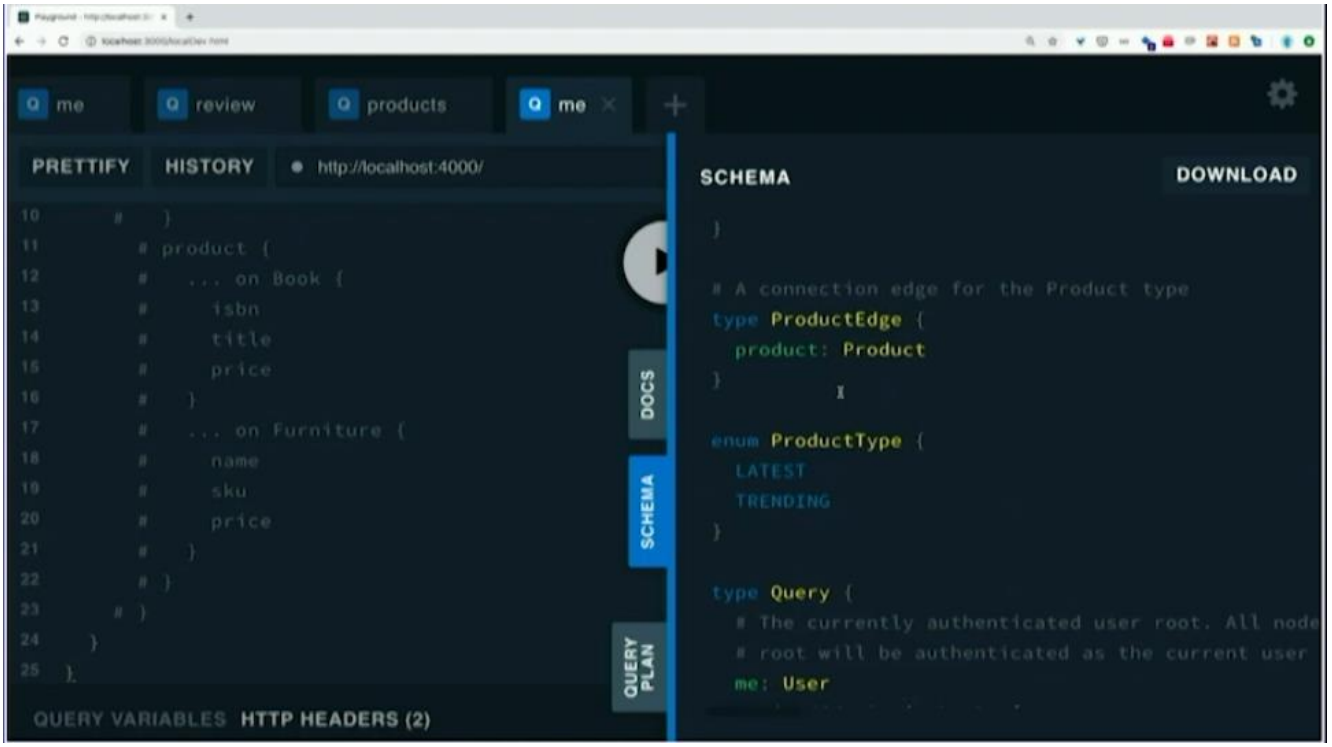

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4002/
COPY CURL
{
review(id: "1") {
author {
id
# username
}
body
id
}
}
{
"data": {
"review": {
"author": {
"id": "1"
},
"body": "Love it!",
"id": "1"
}
}
}
DOCS
SCHEMA

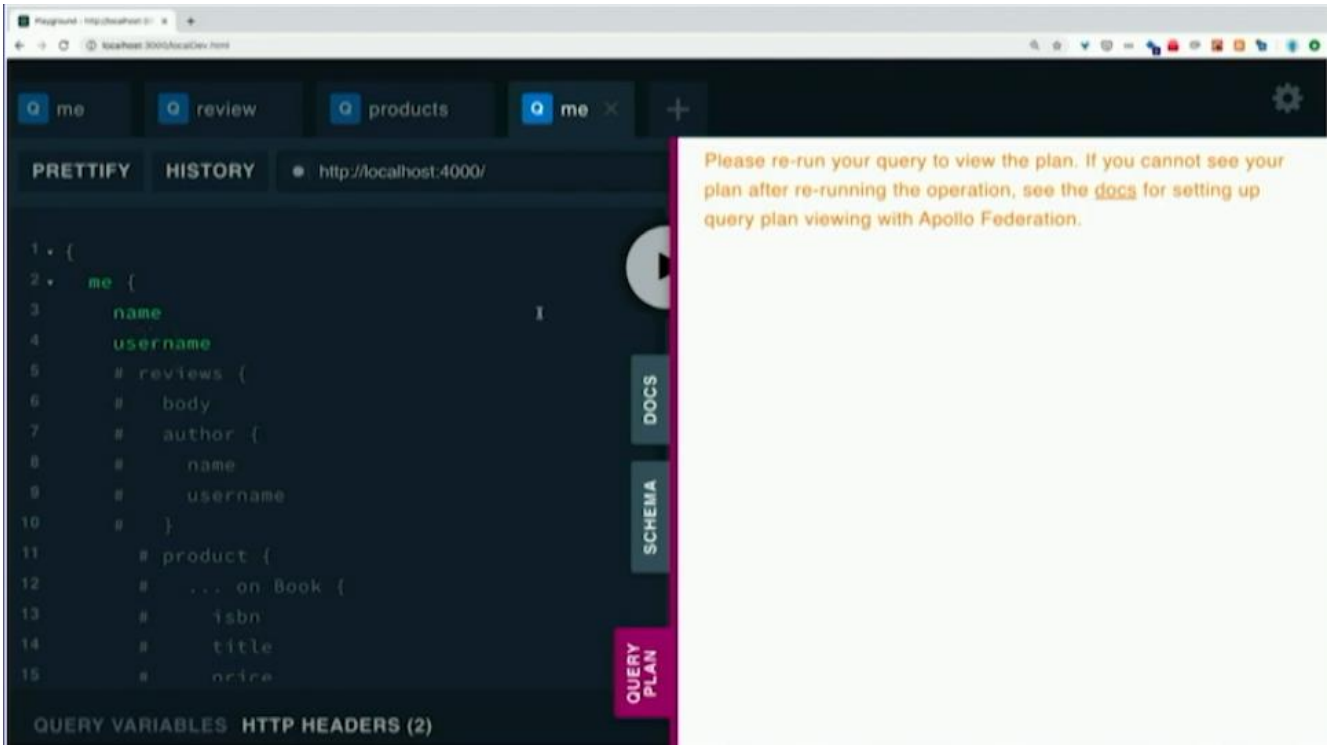

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4002/
COPY CURL
{
review(id: "1") {
author {
id
username
}
body
id
}
}
{
"data": {
"review": {
"author": {
"id": "1",
"username": "@ada"
},
"body": "Love it!",
"id": "1"
}
}
}
DOCS
SCHEMA

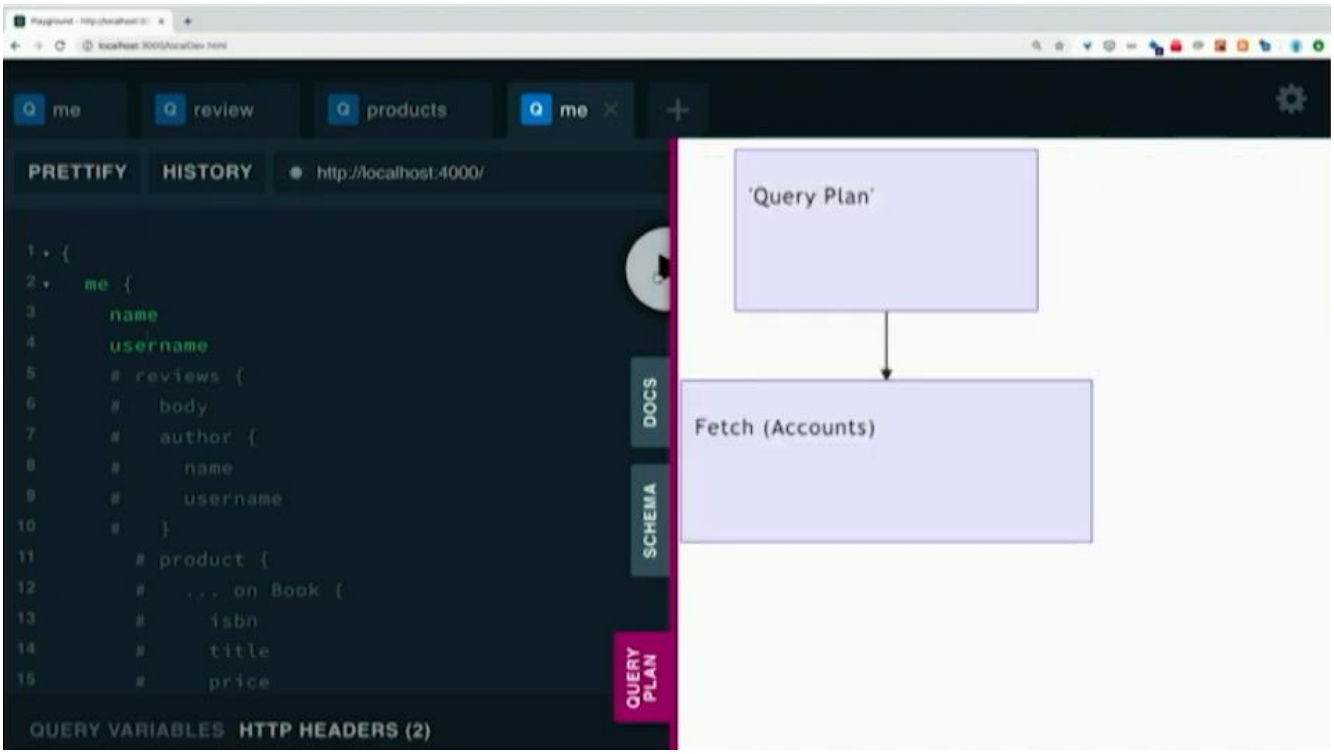

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
SCHEMA
DOWNLOAD
# product {
# ... on Book {
# isbn
# title
# price
# }
# ... on Furniture {
# name
# sku
# price
# }
# }
# }
}
}
# Information about the brand Amazon
type Amazon {
# The url of a referrer for a product
referrer: String
}
# The basic book in the graph
type Book implements Product {
# All books can be found by an isbn
isbn: String!
The title of the book
title: String
The year the book was published
year: Int
A simple list of similar books
DOCS
SCHEMA
QUERY PLAN

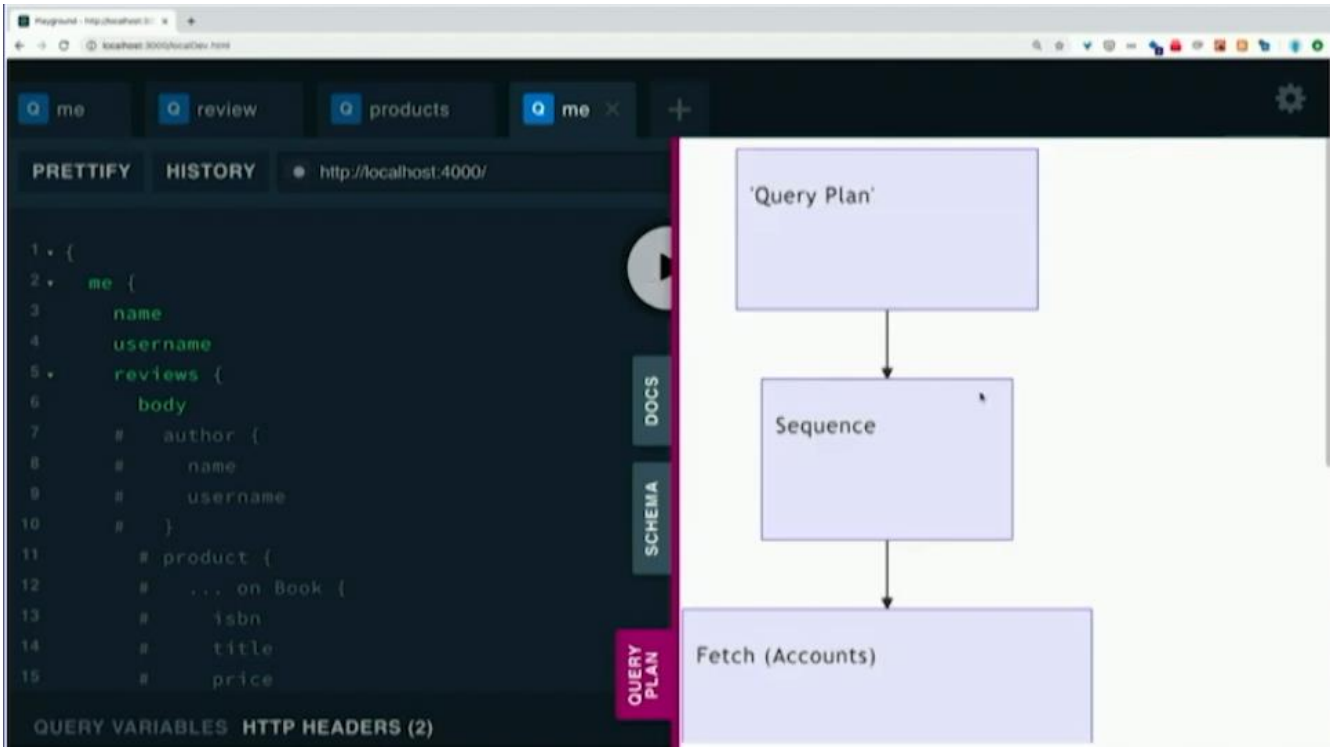

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
SCHEMA
DOWNLOAD
# product {
# ... on Book {
# isbn
# title
# price
# }
# ... on Furniture {
# name
# sku
# price
# }
# }
# }
}
}
# The base User
type User {
# A globally unique id for the user
id: ID!
The users full name as provided
name: String
The account username of the user
username: String
A list of all reviews by the user
reviews: [Review]
}
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)

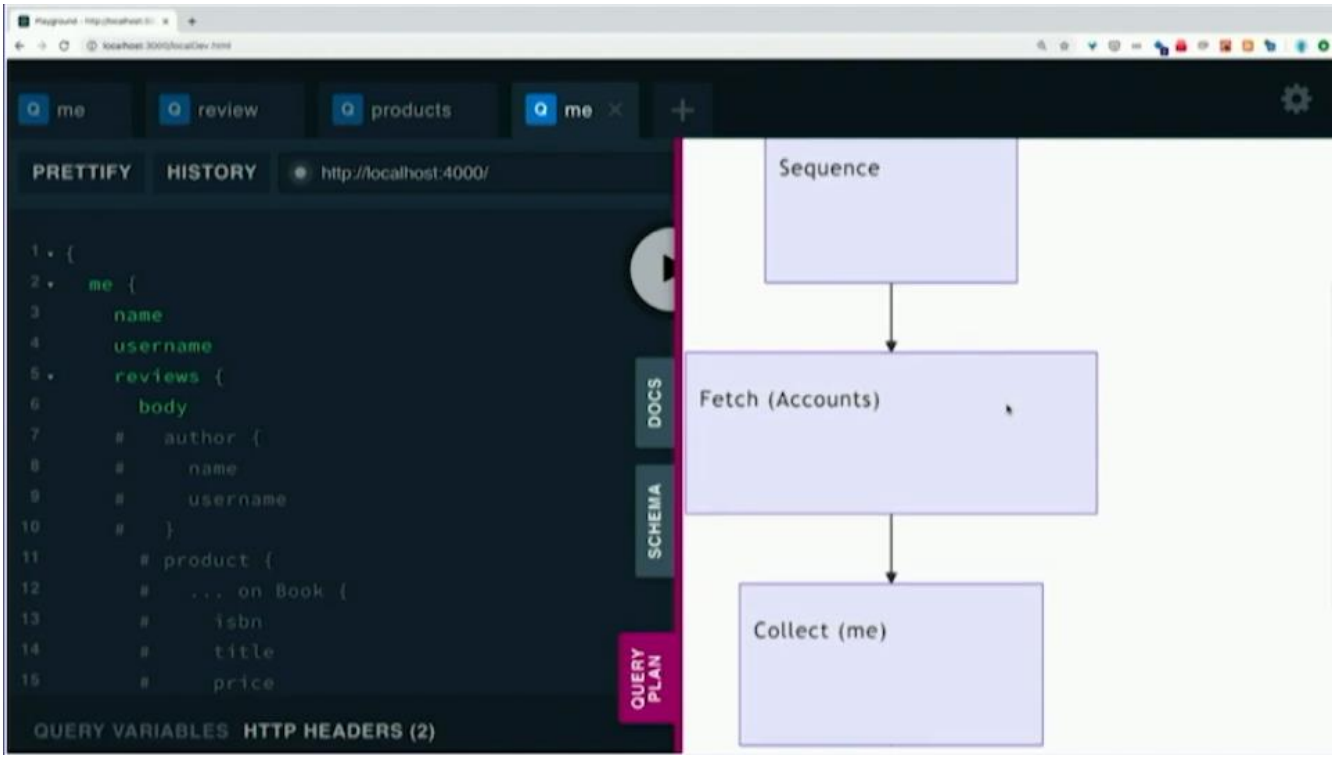

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
SCHEMA
DOWNLOAD
products(first: Int = 5, after: Int = 0, type: Pr
# A review is any feedback about products across th
type Review {
id: ID!
The plain text version of the review
body: String
The user who authored the review
author: User
}
# A connection wrapper for lists of reviews
type ReviewConnection {
# Helpful metadata about the connection
pageInfo: PageInfo
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)

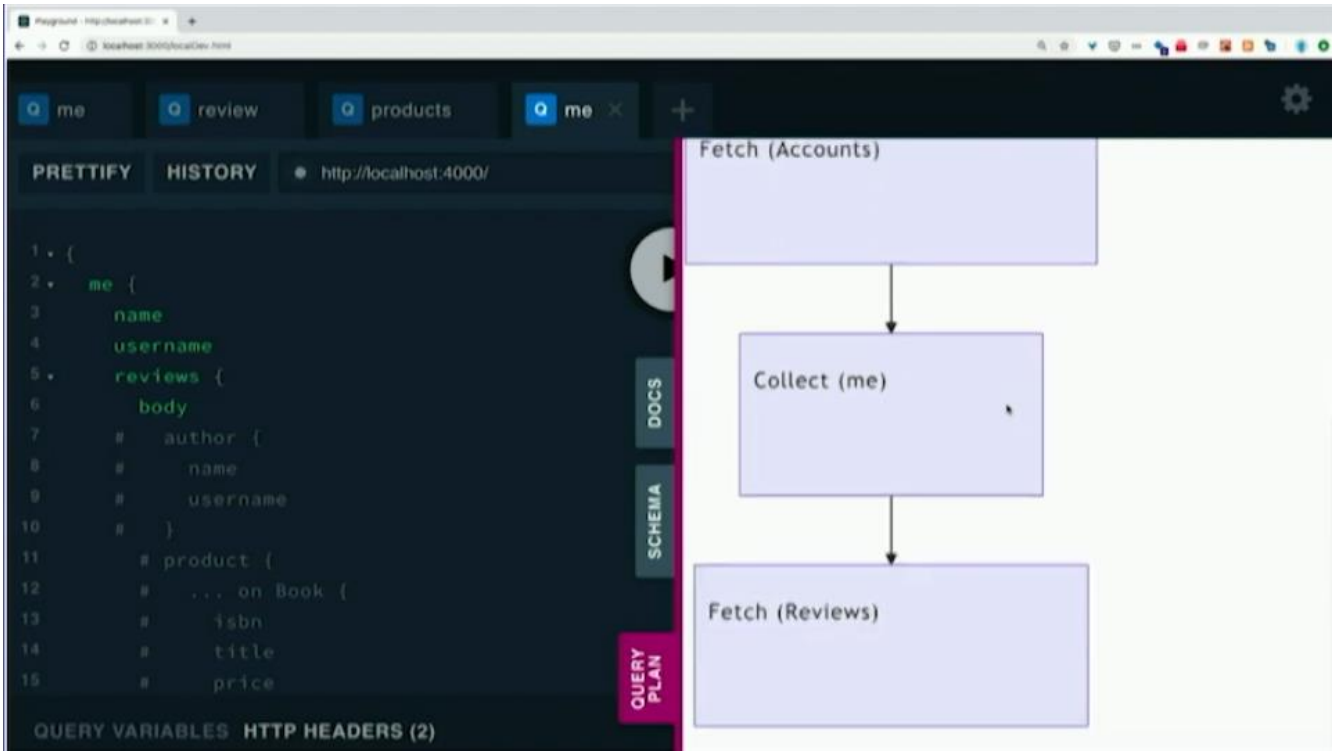

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
SCHEMA
DOWNLOAD
}
# A connection edge for the Product type
type ProductEdge {
product: Product
}
enum ProductType {
LATEST
TRENDING
}
type Query {
# The currently authenticated user root. All node
# root will be authenticated as the current user
me: User
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)

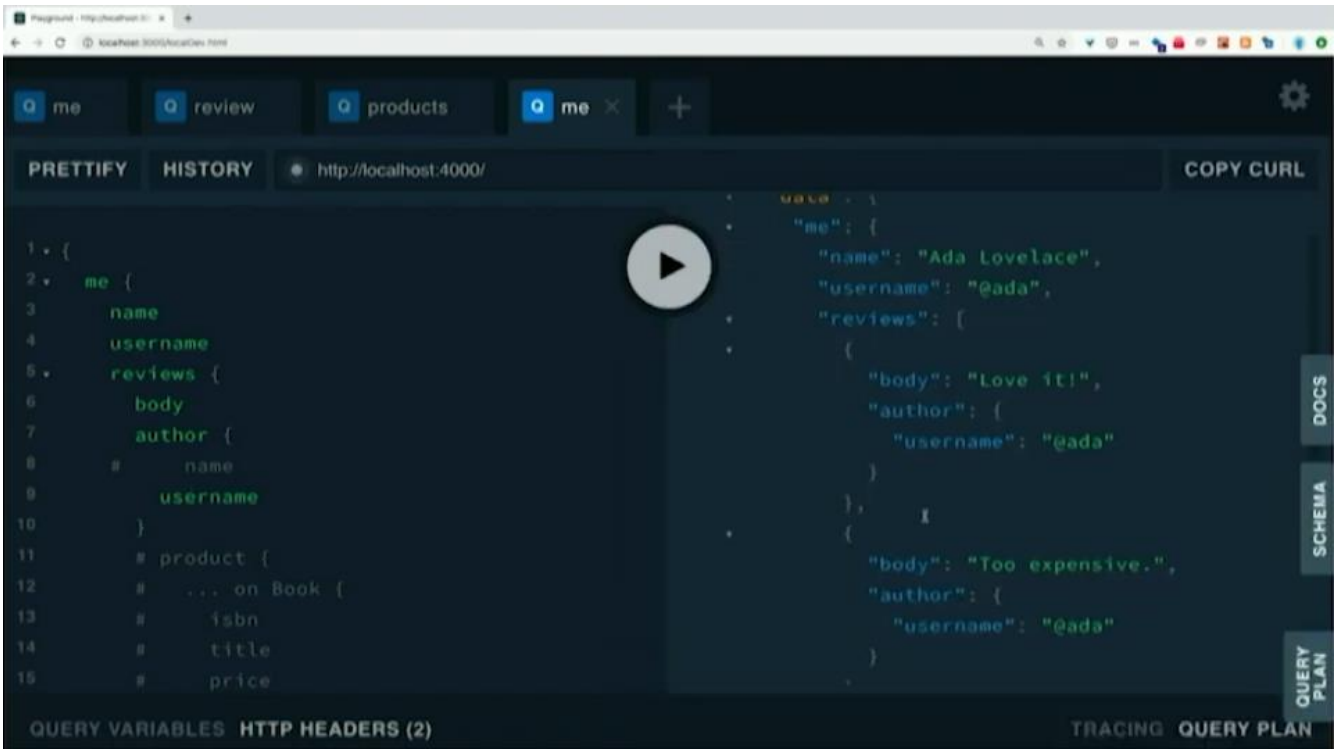

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
name
username
}
product {
... on Book {
isbn
title
price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
Please re-run your query to view the plan. If you cannot see your plan after re-running the operation, see the docs for setting up query plan viewing with Apollo Federation.

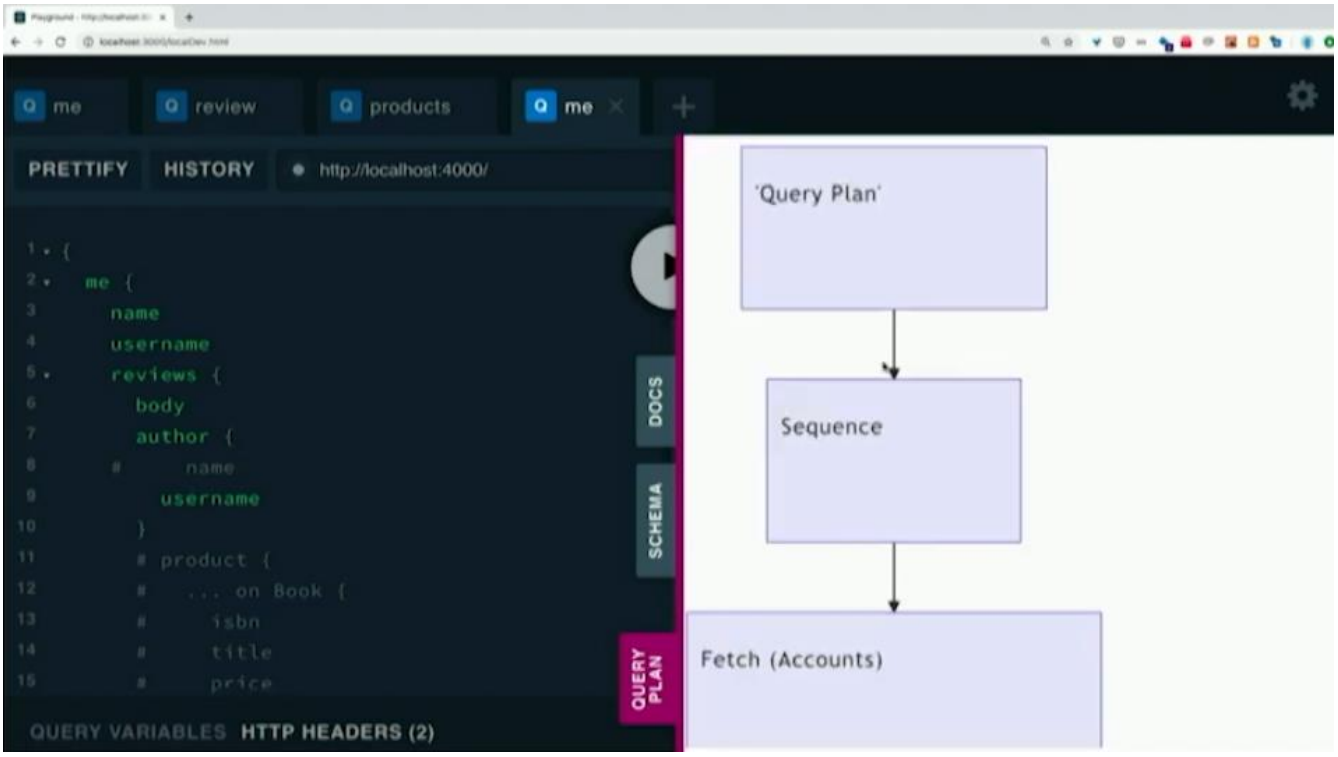

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
name
username
}
product {
... on Book {
isbn
title
price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
'Query Plan'
Fetch (Accounts)

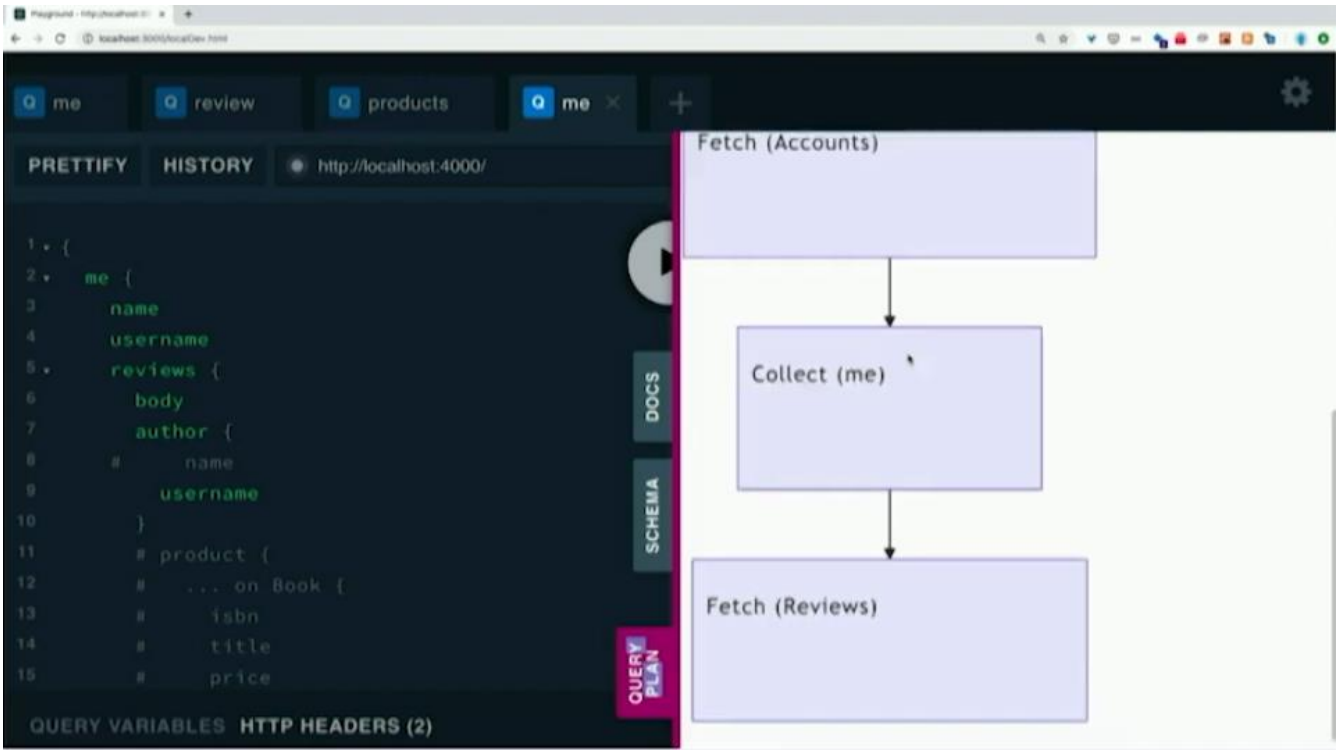

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
name
username
}
product {
... on Book {
isbn
title
price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
'Query Plan'
Sequence
Fetch (Accounts)

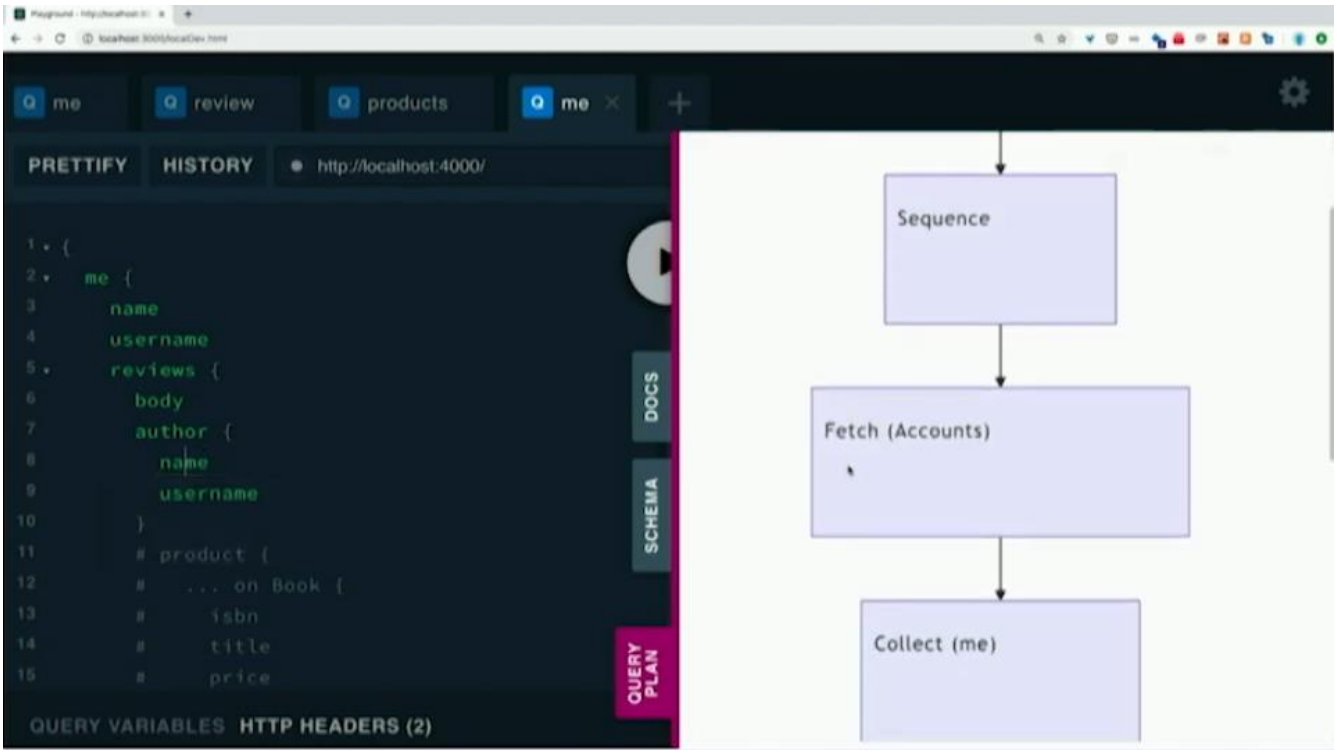

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
name
username
}
product {
... on Book {
isbn
title
price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
Sequence
Fetch (Accounts)
Collect (me)

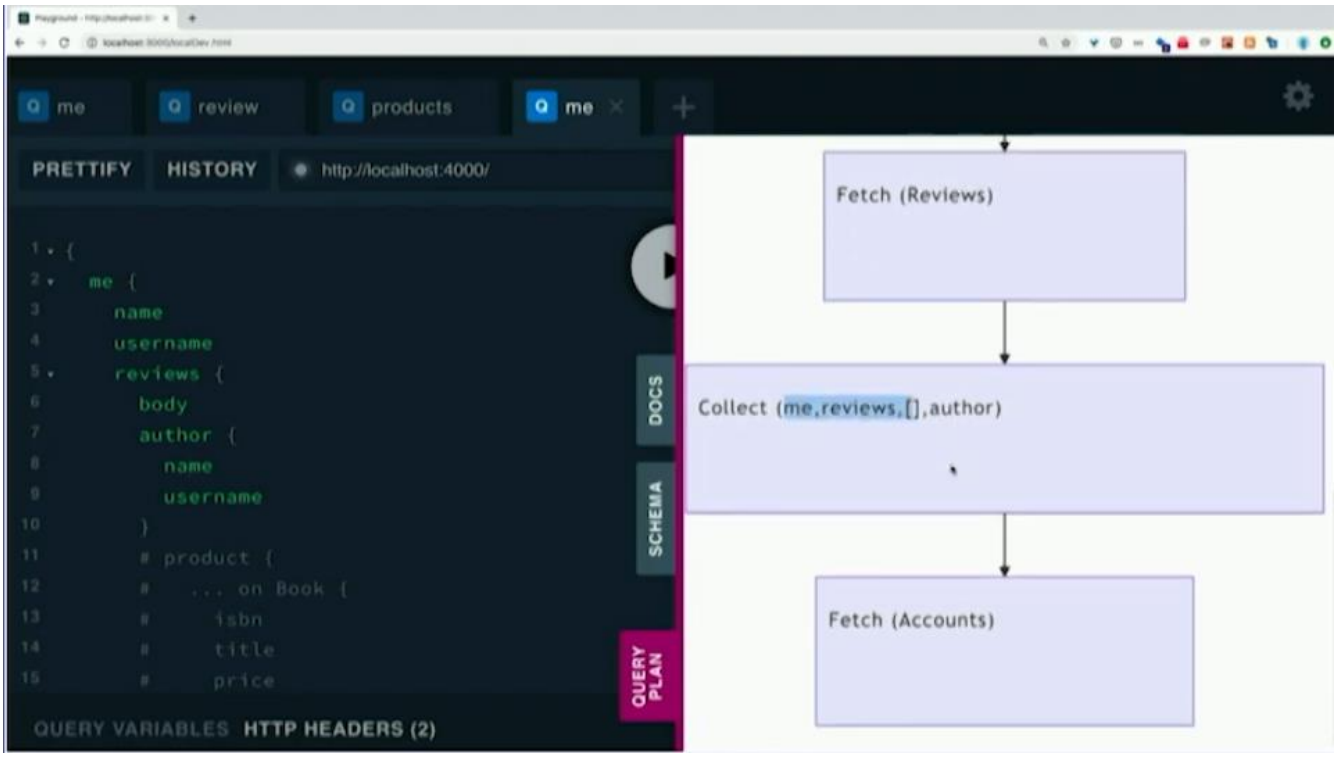

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
name
username
}
product {
... on Book {
isbn
title
price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
Fetch (Accounts)
Collect (me)
Fetch (Reviews)

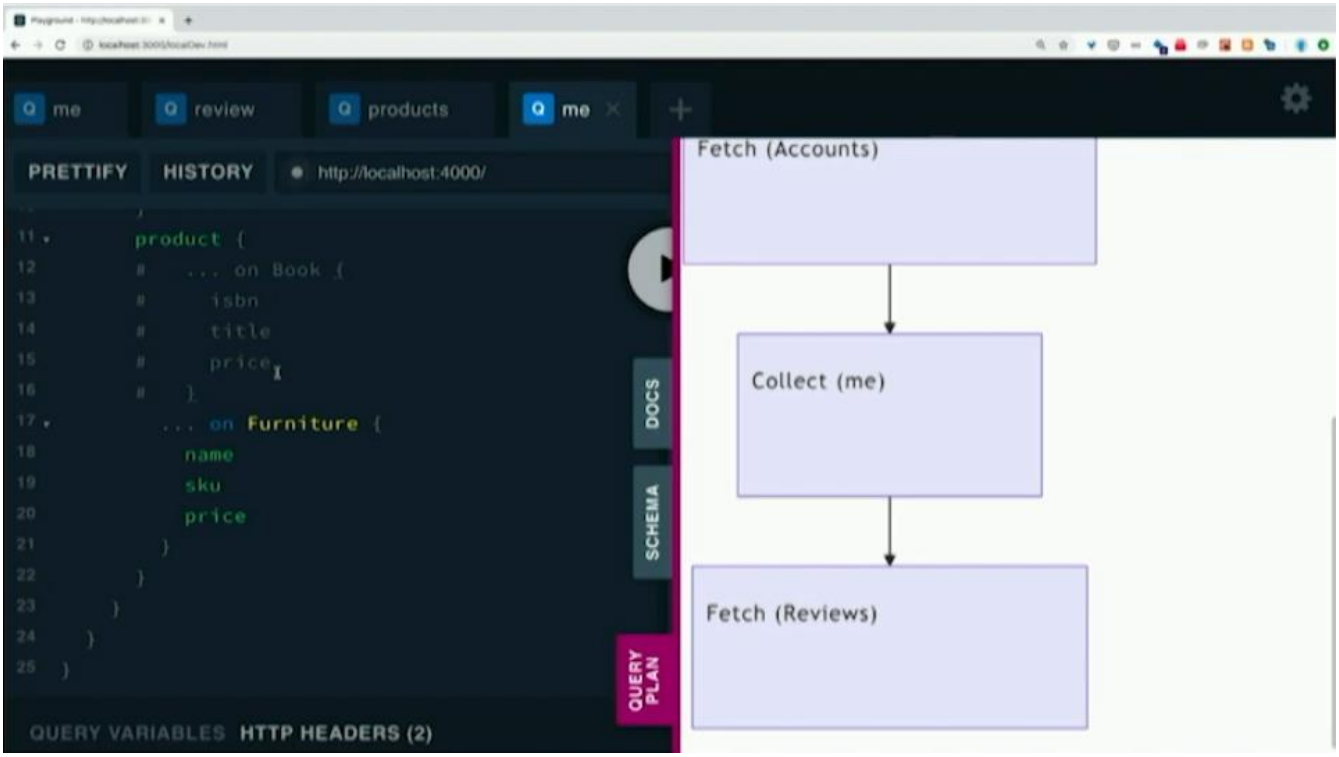

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
COPY CURL
{
me {
name
username
reviews {
body
author {
# name
username
}
# product {
# ... on Book {
# isbn
# title
# price
"me": {
"name": "Ada Lovelace",
"username": "@ada",
"reviews": [
{
"body": "Love it!",
"author": {
"username": "@ada"
}
},
{
"body": "Too expensive.",
"author": {
"username": "@ada"
}
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
TRACING
QUERY PLAN


me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
# name
username
}
# product {
# ... on Book {
# isbn
# title
# price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
'Query Plan'
Sequence
Fetch (Accounts)

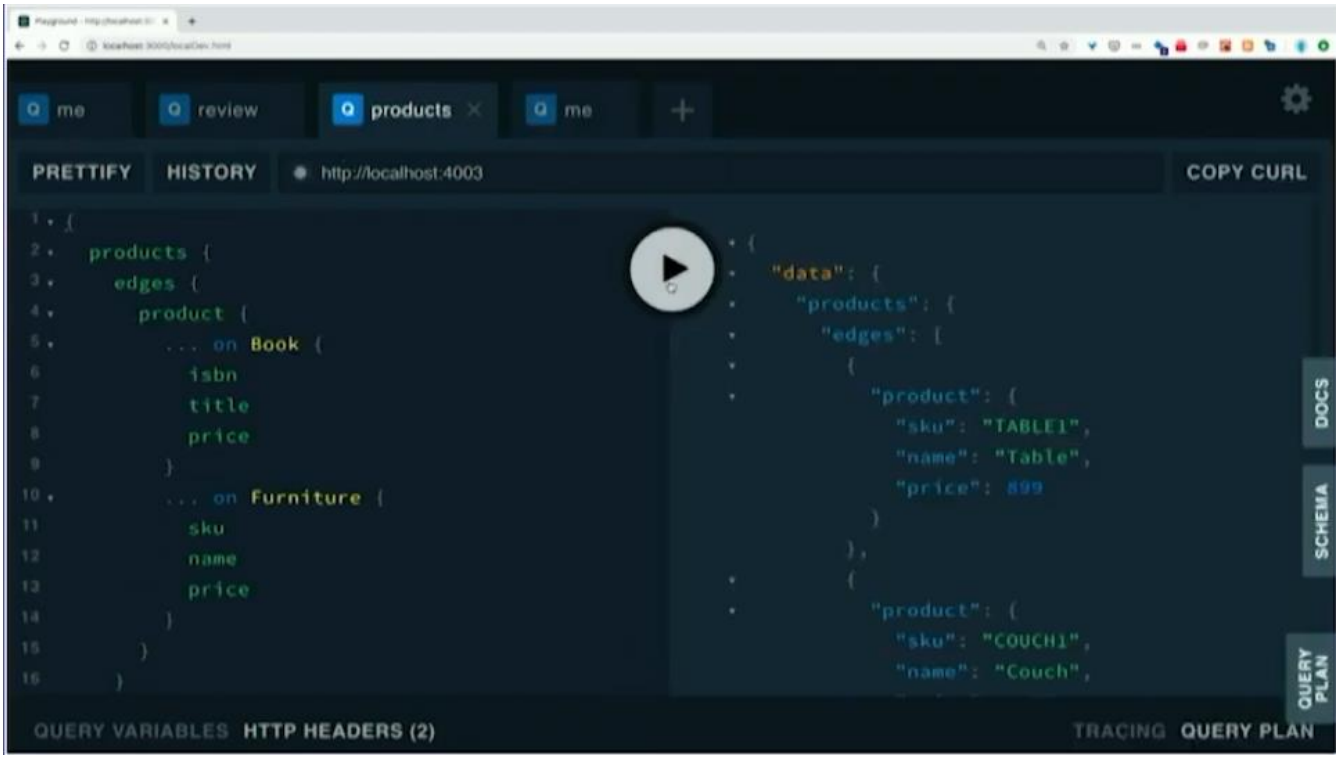

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
# name
username
}
# product {
# ... on Book {
# isbn
# title
# price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
Fetch (Accounts)
Collect (me)
Fetch (Reviews)

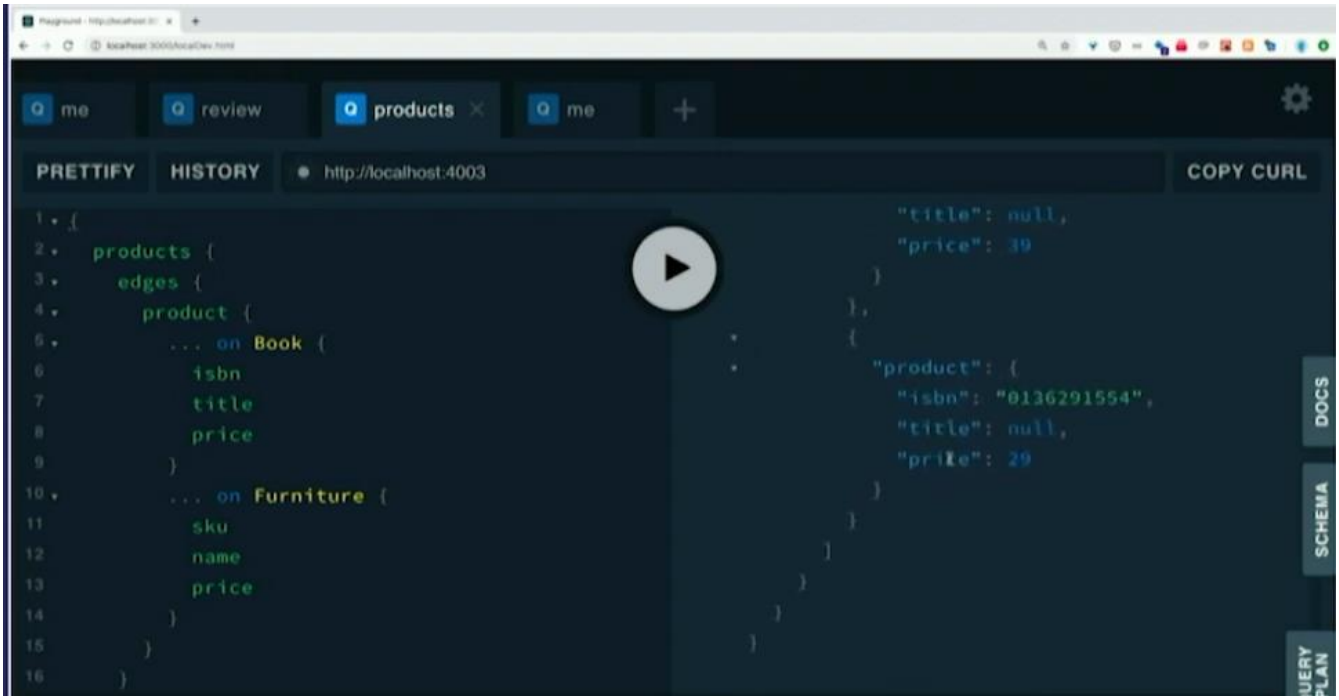

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
name
username
}
# product {
# ... on Book {
# isbn
# title
# price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
Sequence
Fetch (Accounts)
Collect (me)

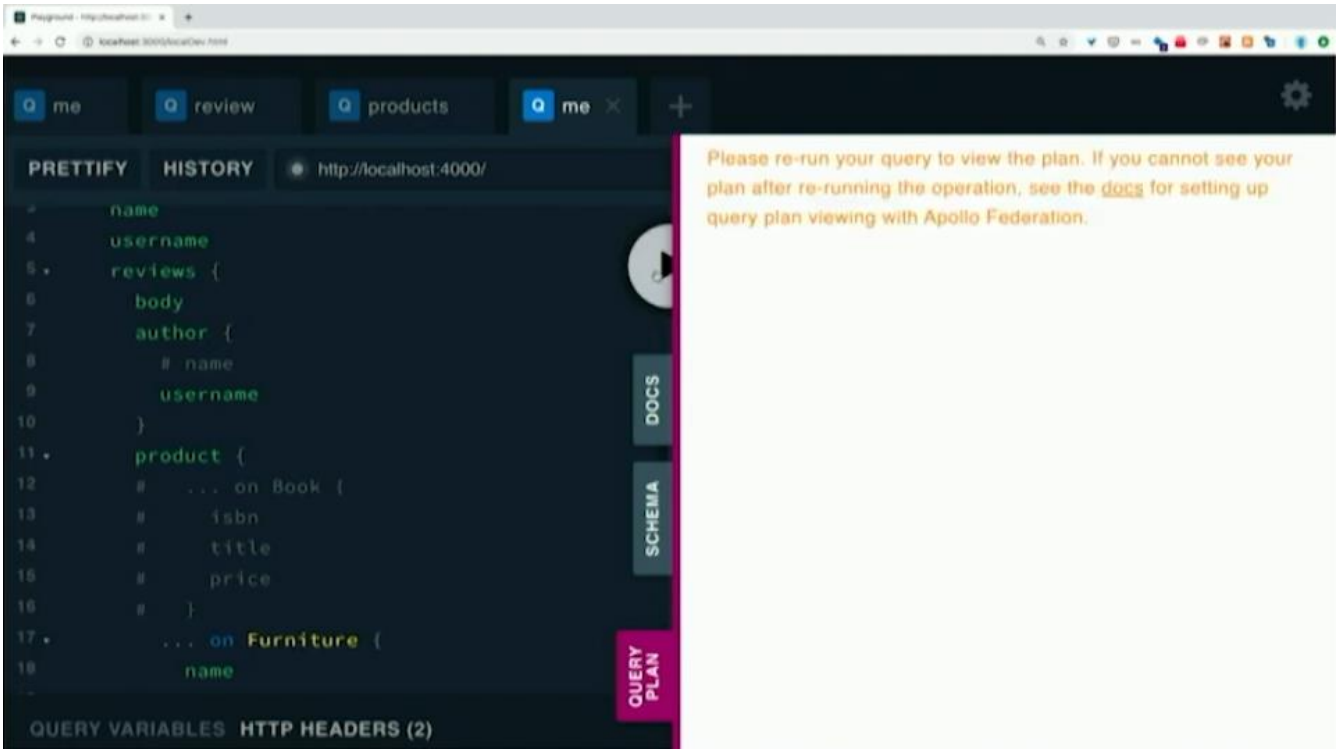

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
{
me {
name
username
reviews {
body
author {
name
username
}
# product {
# ... on Book {
# isbn
# title
# price
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
Fetch (Reviews)
Collect (me,reviews,[],author)
Fetch (Accounts)

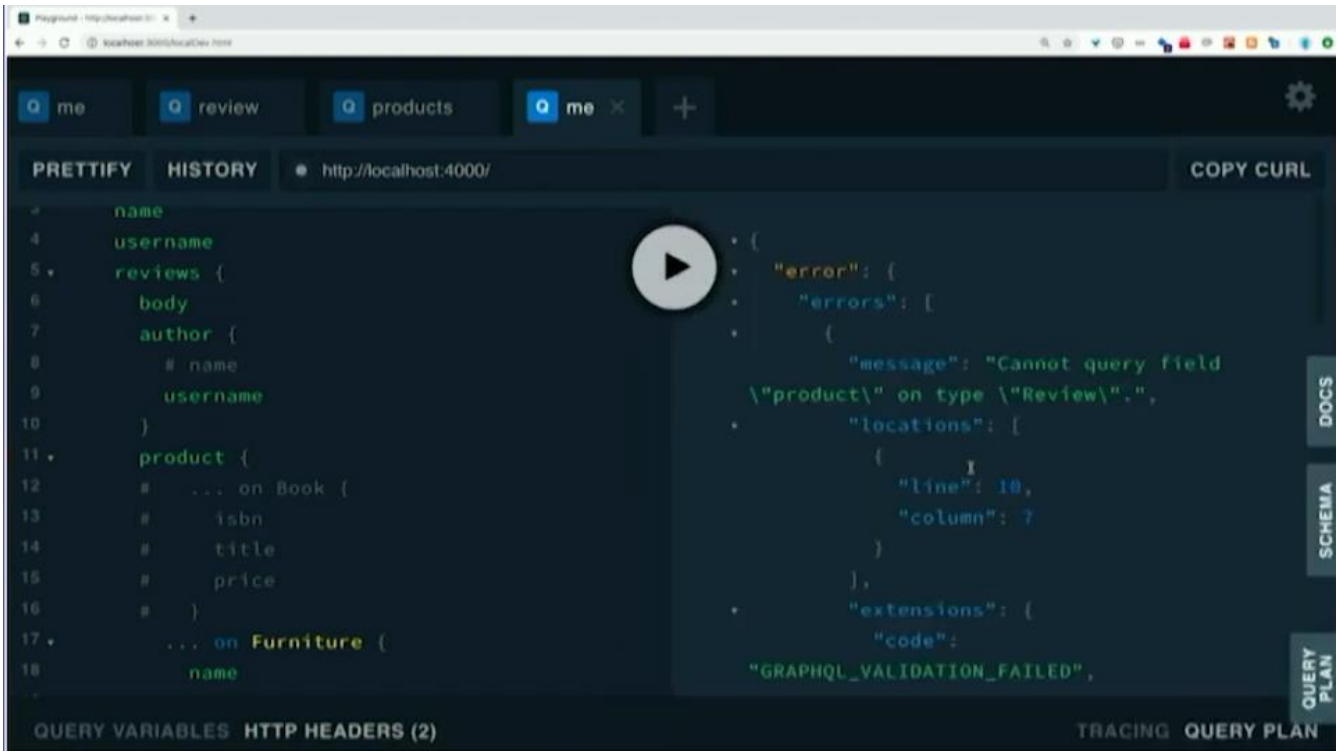


Product is an interface with a Furniture concrete implementation of it

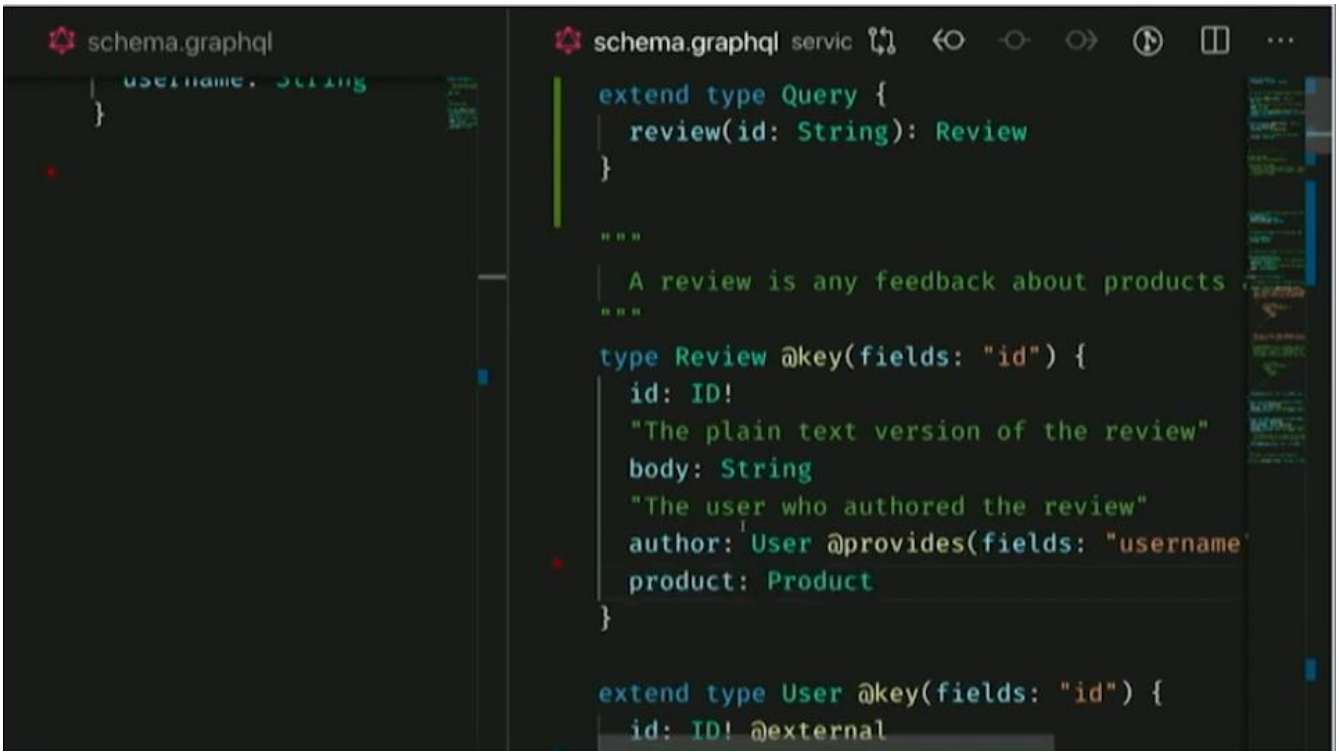


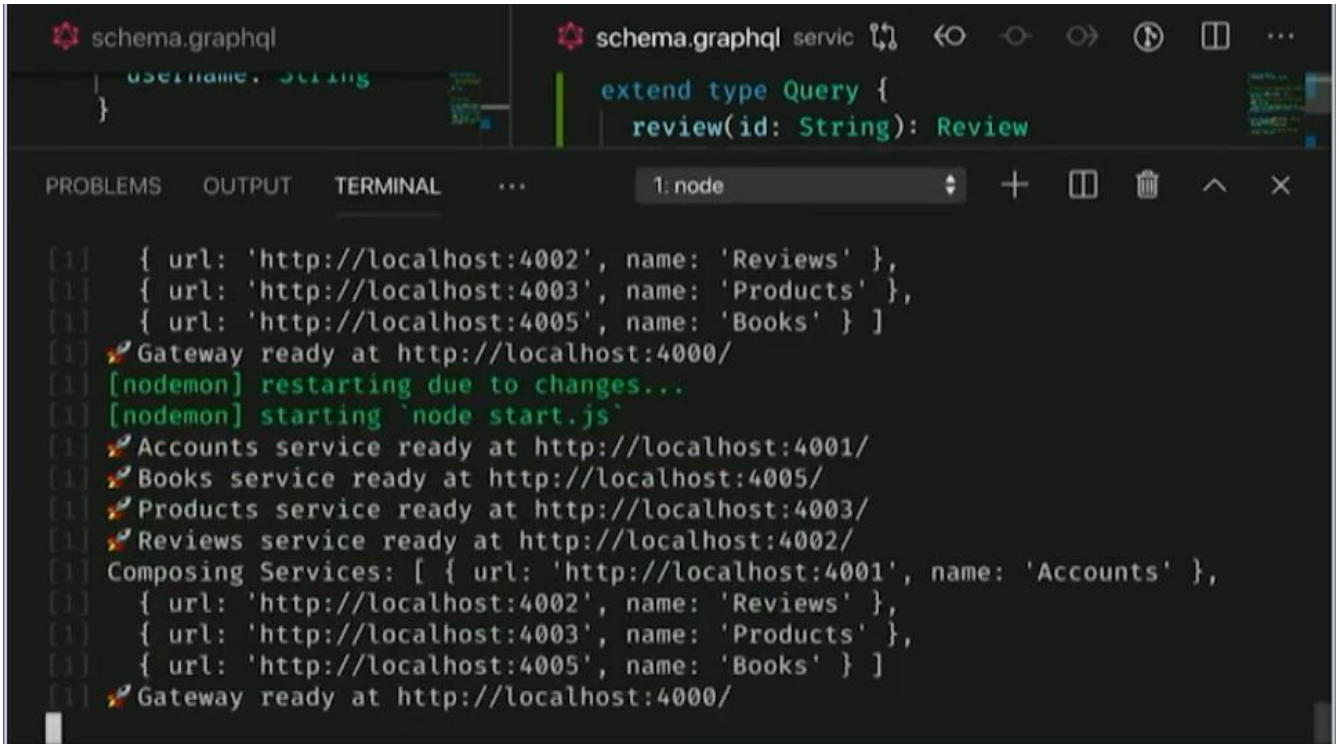

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4003
COPY CURL
{
products {
edges {
product {
... on Book {
isbn
title
price
}
... on Furniture {
sku
name
price
}
}
}
"title": null,
"price": 39
},
{
"product": {
"isbn": "0136291554",
"title": null,
"price": 29
DOCS
SCHEMA

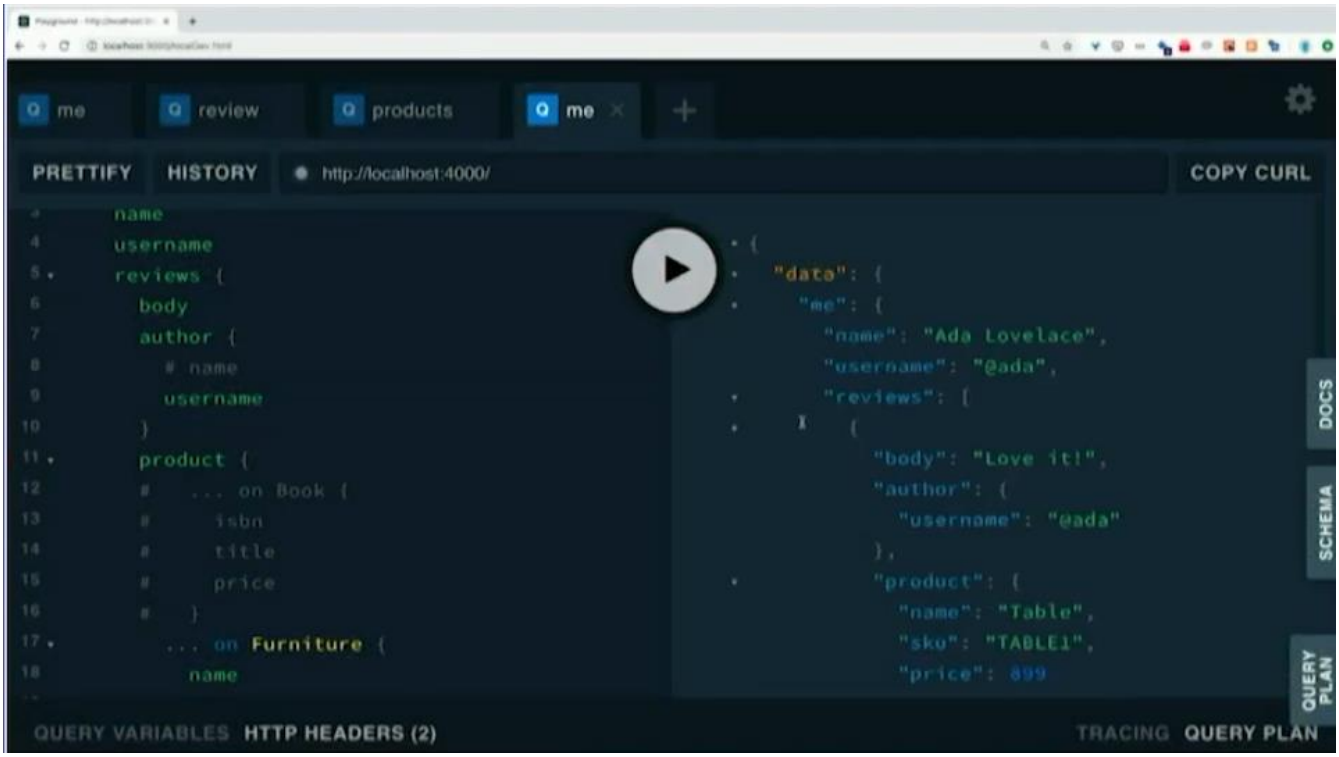

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
name
username
reviews {
body
author {
# name
username
}
product {
# ... on Book {
# isbn
# title
# price
# }
... on Furniture {
name
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
Please re-run your query to view the plan. If you cannot see your plan after re-running the operation, see the docs for setting up query plan viewing with Apollo Federation.

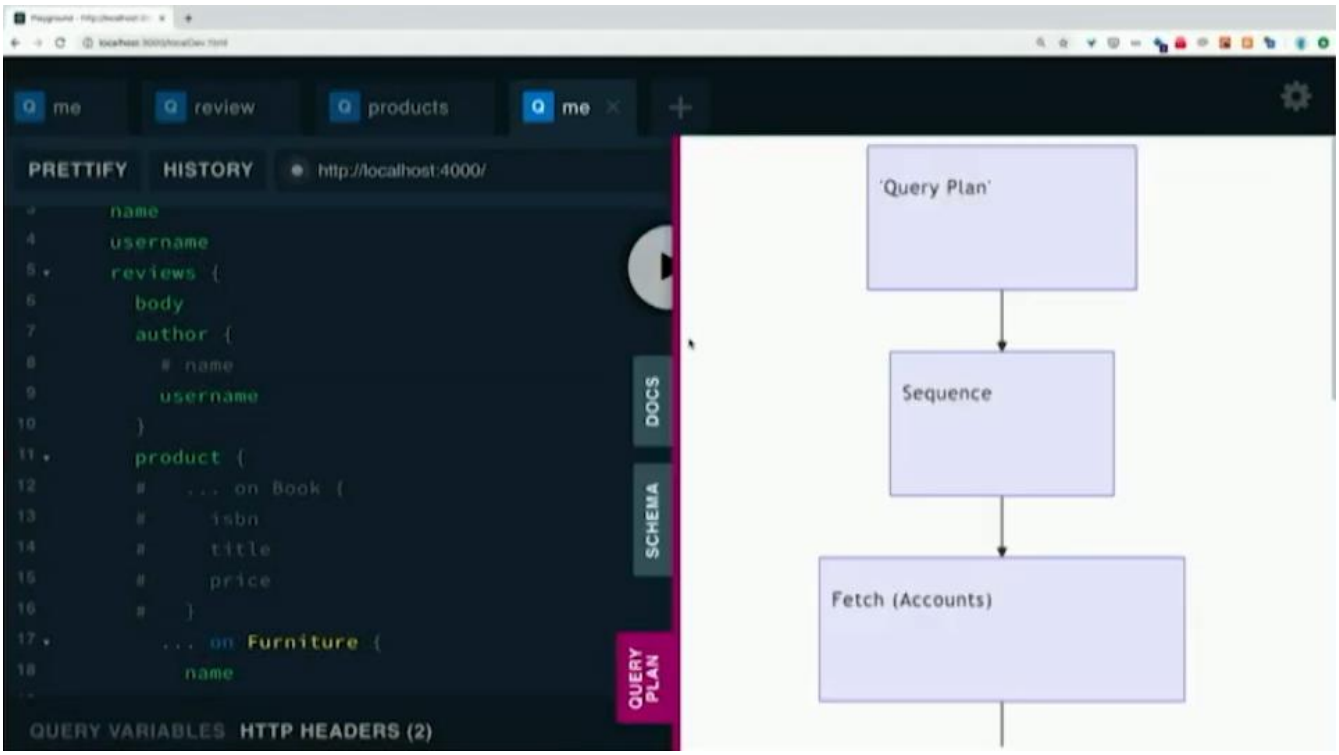


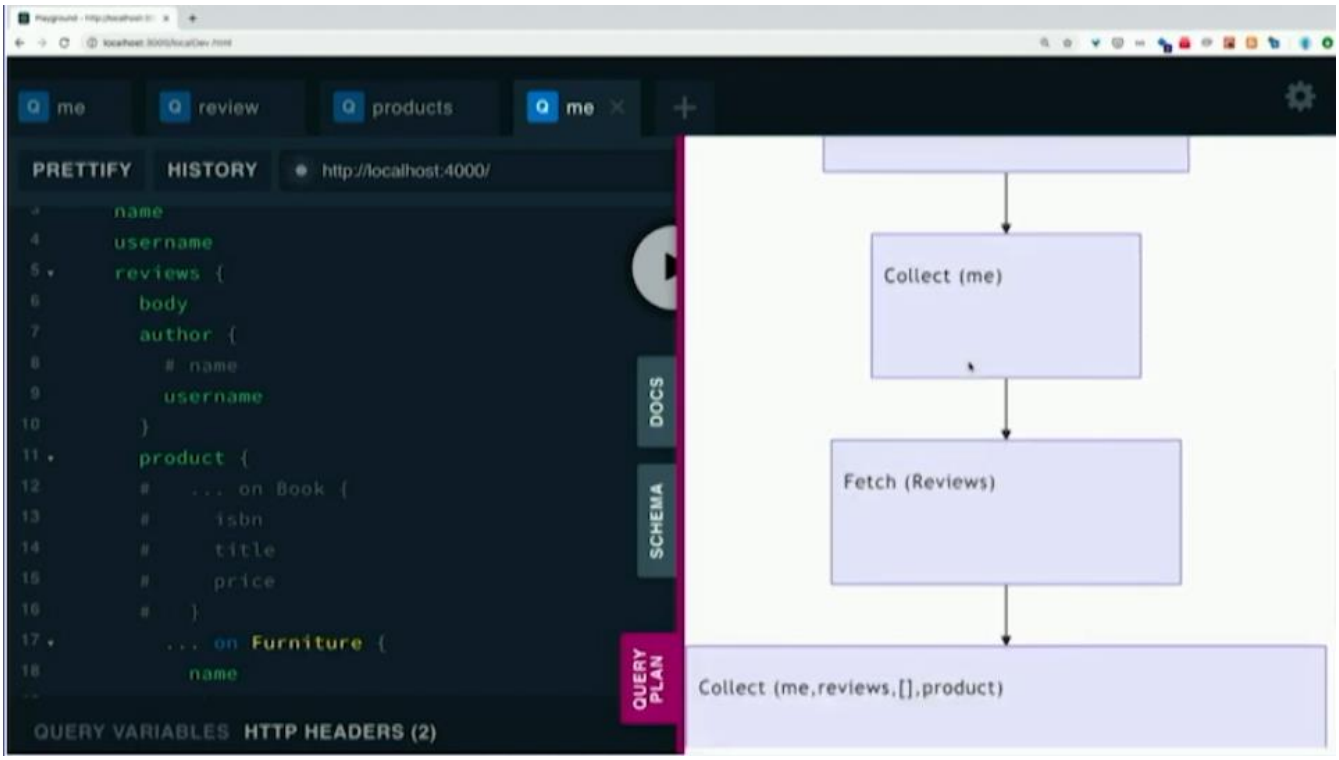
This fails because there is no Product schema available

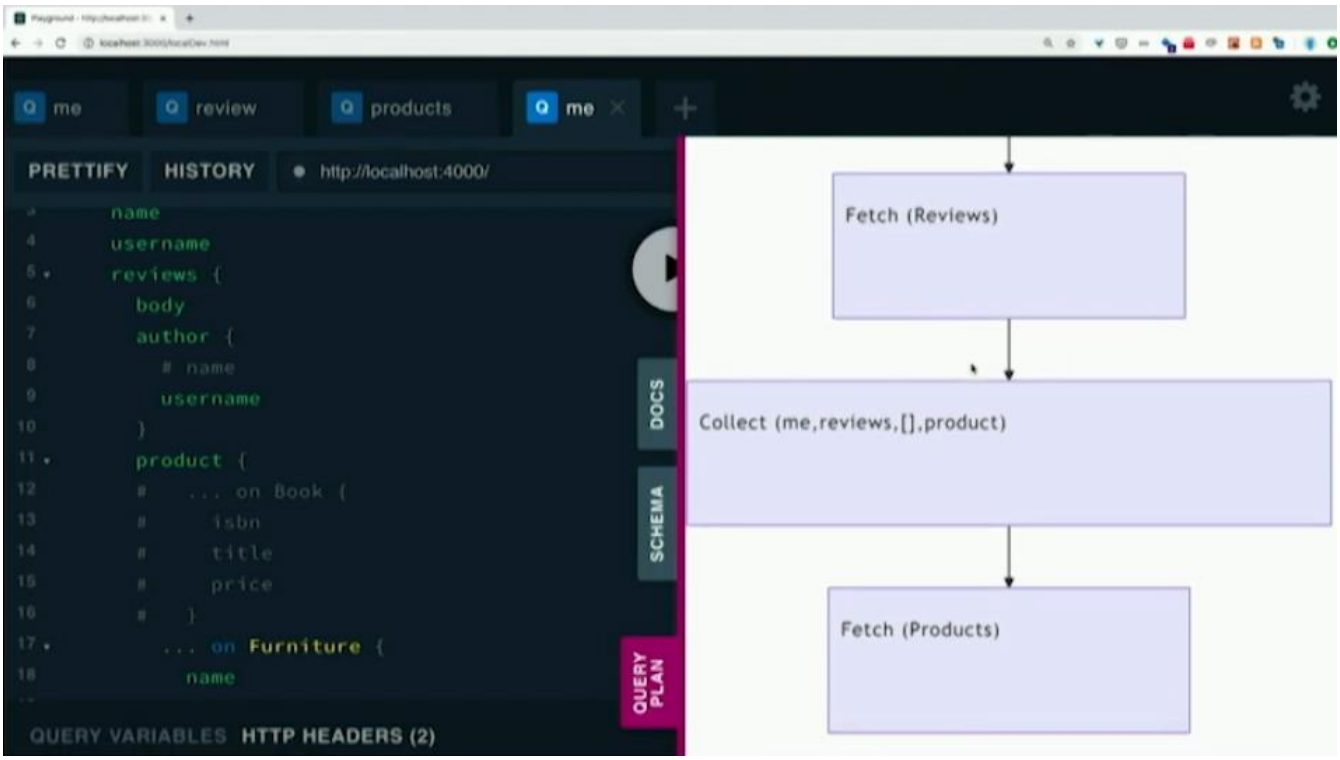



schema.graphql
schema.graphql servic
extend type Query {
review(id: String): Review
PROBLEMS
OUTPUT
TERMINAL
1: node
[1] { url: 'http://localhost:4002', name: 'Reviews' },
[1] { url: 'http://localhost:4003', name: 'Products' },
[1] { url: 'http://localhost:4005', name: 'Books' }]
[1] Gateway ready at http://localhost:4000/
[1] [nodemon] restarting due to changes...
[1] [nodemon] starting `node start.js`
[1] Accounts service ready at http://localhost:4001/
[1] Books service ready at http://localhost:4005/
[1] Products service ready at http://localhost:4003/
[1] Reviews service ready at http://localhost:4002/
[1] Composing Services: [{ url: 'http://localhost:4001', name: 'Accounts' },
[1] { url: 'http://localhost:4002', name: 'Reviews' },
[1] { url: 'http://localhost:4003', name: 'Products' },
[1] { url: 'http://localhost:4005', name: 'Books' }]
[1] Gateway ready at http://localhost:4000/

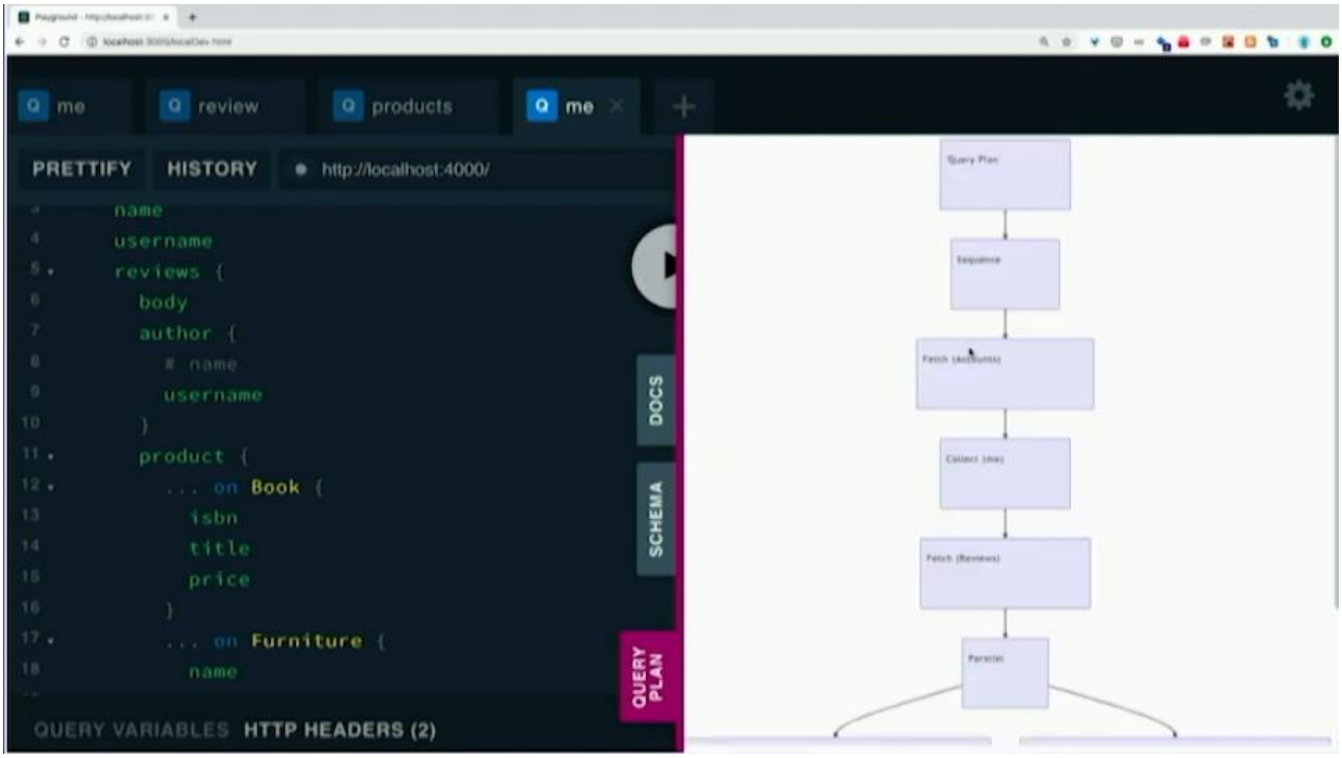

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
COPY CURL
name
username
reviews {
body
author {
# name
username
}
product {
# ... on Book {
# isbn
# title
# price
# }
... on Furniture {
name
"data": {
"me": {
"name": "Ada Lovelace",
"username": "@ada",
"reviews": [
{
"body": "Love it!",
"author": {
"username": "@ada"
},
"product": {
"name": "Table",
"sku": "TABLE1",
"price": 899
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
TRACING
QUERY PLAN


me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
name
username
reviews {
body
author {
# name
username
}
product {
# ... on Book {
# isbn
# title
# price
# }
... on Furniture {
name
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
'Query Plan'
Sequence
Fetch (Accounts)

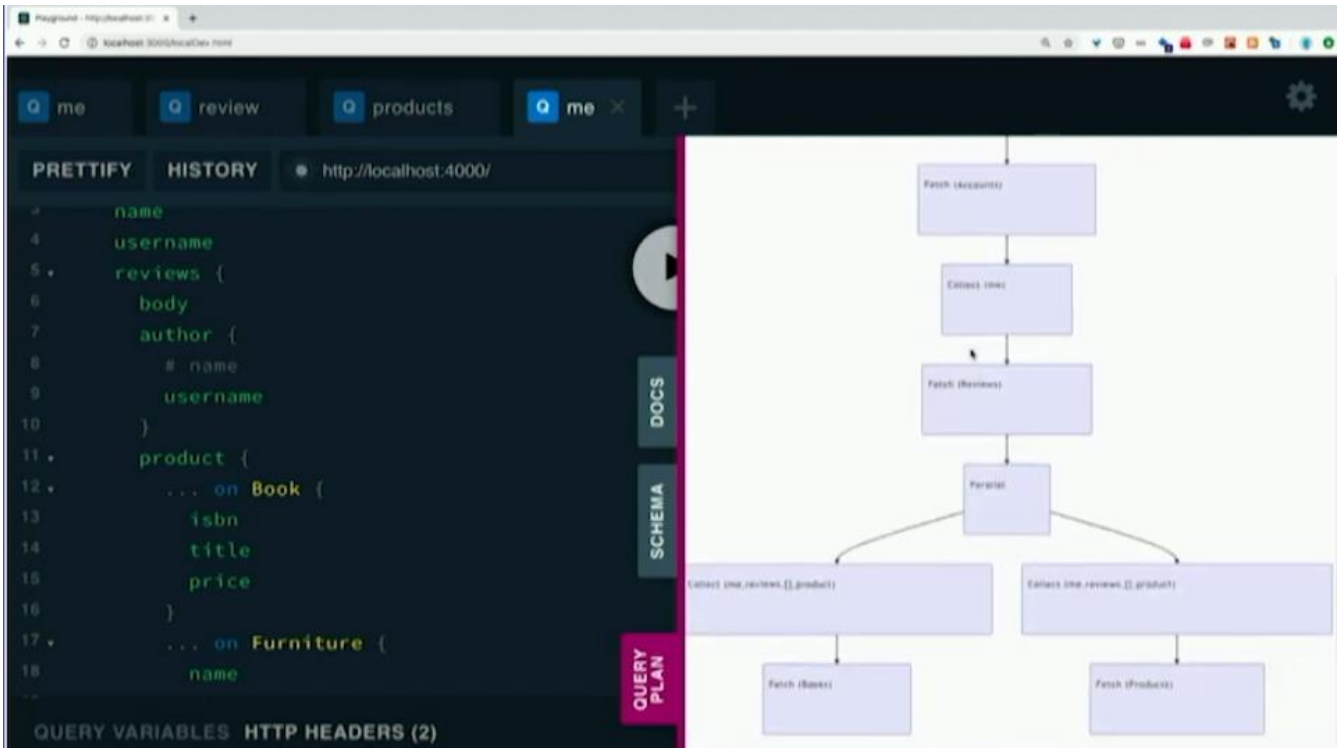

me
review
products
me
PRETTIFY
HISTORY
http://localhost:4000/
name
username
reviews {
body
author {
# name
username
}
product {
# ... on Book {
# isbn
# title
# price
# }
... on Furniture {
name
DOCS
SCHEMA
QUERY PLAN
QUERY VARIABLES
HTTP HEADERS (2)
Collect (me)
Fetch (Reviews)
Collect (me,reviews,[],product)



But the operations we have seen so far are all serial operations, let us see some parallel operations



We have built a dynamic query plan visualizer for this use case that is smart enough to hit 2 services in parallel



DEMO

Entities
Extension
Query Plan
Magic

The Architecture of Federation

A declarative model for Graph composition of loosely coupled downstream GraphQL services that enables static composition and validation of a unified graph using query plans to resolve downstream operations.