NILS HARTMANN

Schema Stitching with Apollo GraphQL

Slides: https://bit.ly/nordic-coding-graphql

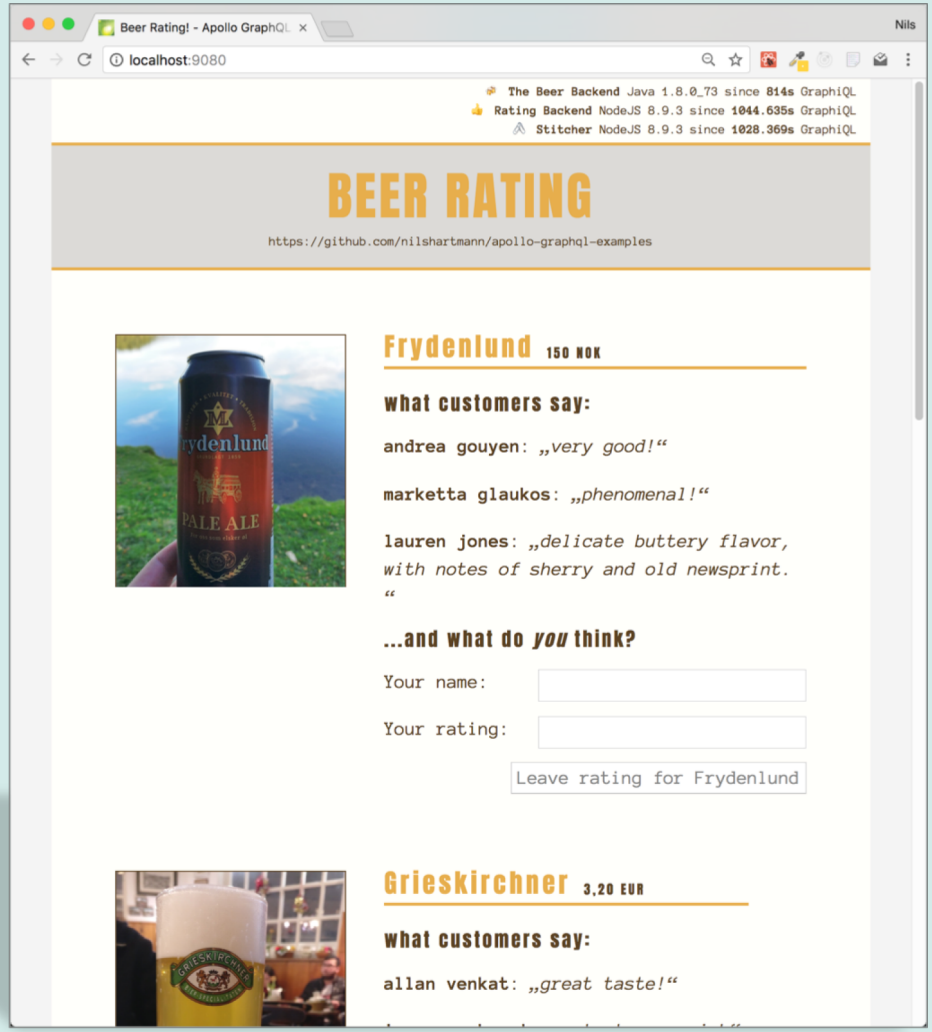
NILS HARTMANN

Software Developer from Hamburg

JavaScript, TypeScript, React
Java

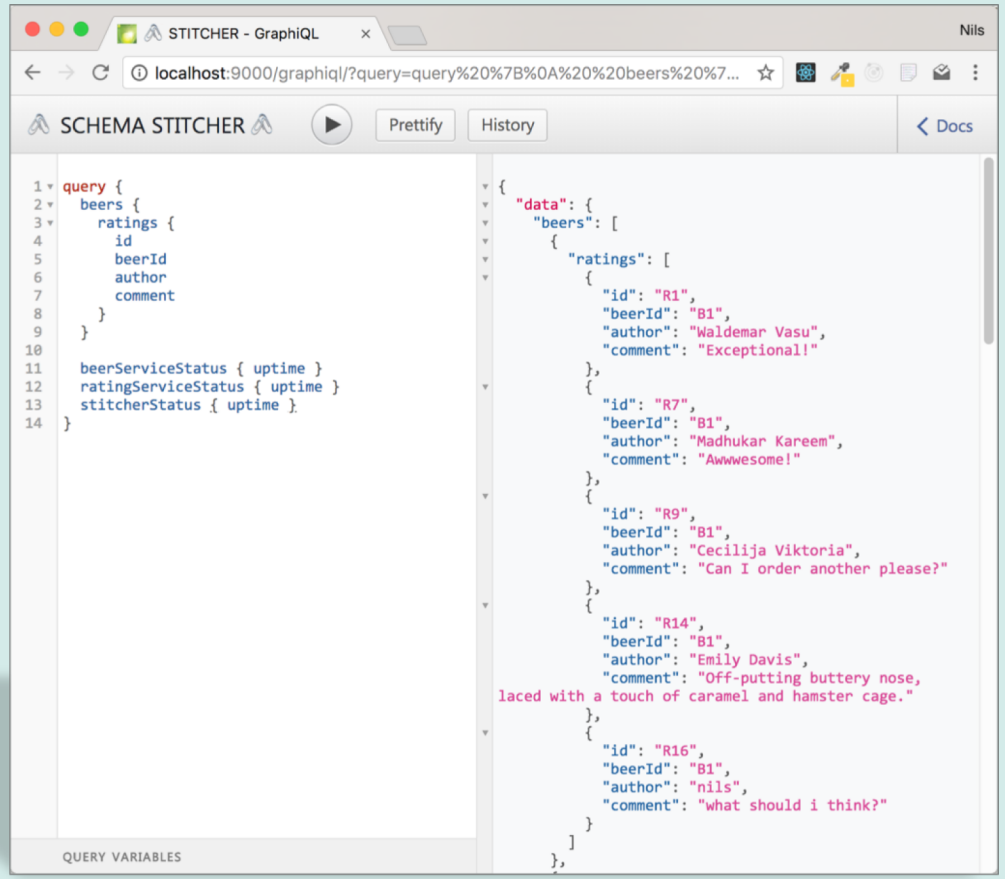
Trainings, Workshops

nils@nilshartmann.net



Example

Source Code: https://bit.ly/graphql-stitching-example



Demo: GraphiQL

http://localhost:9000

APOLLO GRAPHQL

Apollo Server: https://www.apollographql.com/docs/apollo-server/

- Open-Source-Framework for building GraphQL Server (JavaScript)
- Exposes Types and Resolver (Schema) via HTTP endpoint

Example: A GraphQL Server with Apollo

```
// imports omitted

const schema = makeExecutableSchema({
  typeDefs: `Query { hello { name: String } }`,
  resolvers: {
    Query: { hello: () => ({ name: "World" }) }
  }
});

const app = express();
app.use("/graphql", bodyParser.json(), graphqlExpress({ schema }));

app.listen();
```

OVERVIEW

Tasks

1. Fetch the individual schemas from their remote endpoint
2. Rename the `ProcessInfo` type from each schema
3. Create the "link" between the two schemas
    - Add `ratings` field to `Beer` type
4. Merge schemas into one new Schema
5. Expose merged Schema via HTTP Endpoint

Step 1: Fetch the Remote Schemas

```
import fetch from "node-fetch";
import { introspectSchema, makeRemoteExecutableSchema } from "graphql-tools";
import { HttpLink } from "apollo-link-http";

async function createRemoteSchema(uri) {
  // Configure network connection
  const link = new HttpLink({ uri, fetch });

  // Fetch the schema
  const schema = await introspectSchema(link);

  // create an "executable" schema
  return makeRemoteExecutableSchema({
    schema,
    link
  });
}

const beerSchema = await createRemoteSchema("localhost:9010/graphql");
const ratingSchema = await createRemoteSchema("localhost:9020/graphql");
```

TRANSFORMING SCHEMAS

Step 2: Rename Types and Root-Fields

1. Type `ProcessInfo` to `{System}Status`

Beer

```
type ProcessInfo {
  name: String!
  javaVersion: String!
}
```

```
type BeerStatus {
  name: String!
  javaVersion: String!
}
```

Rating

```
type ProcessInfo {
  name: String!
  nodeJsVersion: String!
}
```

```
type RatingStatus {
  name: String!
  nodeJsVersion: String!
}
```

TRANSFORMING SCHEMAS

Step 2: Rename Types and Root-Fields

1. Type `ProcessInfo` to `{System}Status`

2. Root-Field `ping` to `{system}Status`

Beer

```
type ProcessInfo {
  name: String!
  javaVersion: String!
}

Query {
  beers: [Beer!]!
  ping: ProcessInfo!
}
```

```
type BeerStatus {
  name: String!
  javaVersion: String!
}

Query {
  beers: [Beer!]!
  beerStatus: BeerStatus!
}
```

Rating

```
type ProcessInfo {
  name: String!
  nodeJsVersion: String!
}

Query {
  ratings: [Rating!]!
  ping: ProcessInfo!
}
```

```
type RatingStatus {
  name: String!
  nodeJsVersion: String!
}

Query {
  ratings: [Rating!]!
  ratingStatus: RatingStatus!
}
```

TRANSFORMING SCHEMAS

Step 2: Rename Types and Root-Fields

- `transformSchema` expects a List of Transform operations
- Returns new Schema instance

```
import { transformSchema                                    } from "graphql-tools";

function renameInSchema(schema) {
  return transformSchema(schema, [









  ]);
}
```

TRANSFORMING SCHEMAS

Rename Type `ProcessInfo` to `{System}Status`

```
import { transformSchema, RenameTypes                    } from "graphql-tools";

function renameInSchema(schema, systemName) {
  return transformSchema(schema, [
    // 1. ProcessInfo => XyzStatus
    new RenameTypes(
      name => (name === "ProcessInfo" ? `${systemName}Status` : name




  ]);
}
```

Rename Root-Field `ping` to {`system`}`Status`

```
import { transformSchema, RenameTypes, RenameRootFields } from "graphql-tools";

function renameInSchema(schema, systemName) {
  return transformSchema(schema, [
    // 1. ProcessInfo => XyzProcessStatus
    new RenameTypes(
      name => (name === "ProcessInfo" ? `${systemName}Status` : name
    ),

    // 2. ping => xyzStatus
    new RenameRootFields(
      (op, name) => name === "ping" ? : `${systemName}Status` : name
    )
  ]);
}
```

Run the transformation in both Schemas

```
import { transformSchema, RenameTypes, RenameRootFields } from "graphql-tools";

function renameInSchema(schema, systemName) {
  return transformSchema(schema, [
    // 1.
    new RenameTypes(
      name => (name === "ProcessInfo" ? `${systemName}Status` : name
    ),

    // 2.
    new RenameRootFields(
      (op, name) => name === "ping" ? : `${systemName}Status` : name
    )
  ]);
}


const renamedBeerSchema = renameInSchema(beerSchema, "Beer");
const renamedRatingSchema = renameInSchema(ratingSchema, "Rating");
```

LINK SCHEMAS

Enhance existing schema

1. Add `ratings` to Beer Schema

Beer

```
type Beer {
  id: ID!
  name: String!
  price: String

}
```

```
type Beer {
  id: ID!
  name: String!
  price: String!
  ratings: [Rating!]!
}
```

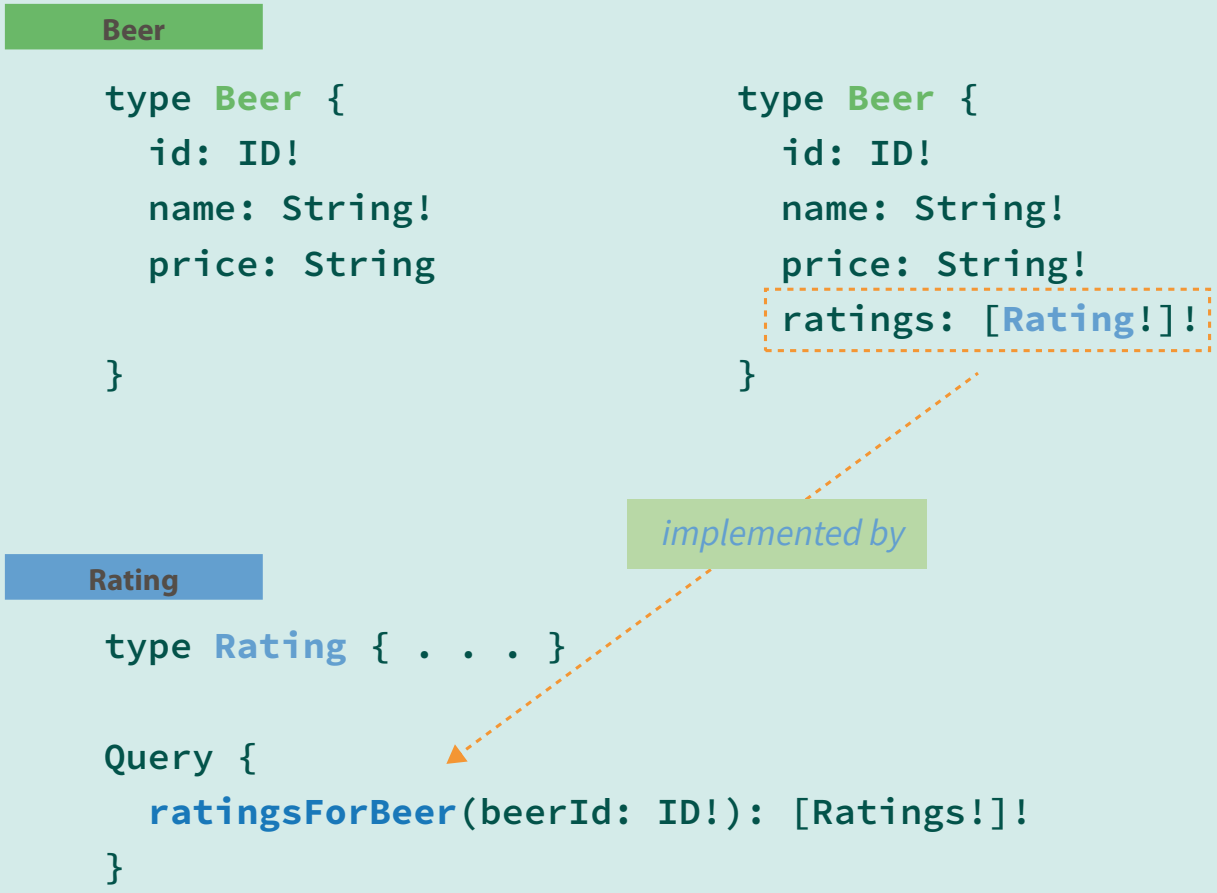
Rating

```
type Rating { . . . }
```

LINK SCHEMAS

Enhance existing schema

1. Add **`ratings`** to Beer Schema
2. Delegate **`requests`** to ratings to Beer Endpoint



LINK SCHEMAS

Extend existing Type

- Add `ratings` field to Beer Schema

```
function createLinkedSchema() {
  return {
    linkedTypeDefs: `
      extend type Beer {
        ratings: [Rating!]!
      }
    `
  }
}
```

LINK SCHEMAS

Add Resolver

Namingconvention as in "regular" Apollo Resolvers

```
function createLinkedSchema(ratingSchema) {
  return {
    linkedTypeDefs: `extend type Beer {ratings: [Rating!]! }`,
    linkedResolvers: {
      Beer: {
        ratings: {
        }
      }
    }
  }
}
```

LINK SCHEMAS

Add Resolver

- Add Fragment to add fields needed for resolver from remote query

```
function createLinkedSchema(ratingSchema) {
  return {
    linkedTypeDefs: `extend type Beer {ratings: [Rating!]! }`,
    linkedResolvers: {
      Beer: {
        ratings: {
          fragment: `fragment BeerFragment on Beer { id }`,
        }
      }
    }
  }
}
```

LINK SCHEMAS

Add Resolver

Delegate execution to the other schema ("Rating" in our example)

```
function createLinkedSchema(ratingSchema) {
  return {
    linkedTypeDefs: `extend type Beer {ratings: [Rating!]! }`,
    linkedResolvers: {
      Beer: {
        ratings: {
          fragment: `fragment BeerFragment on Beer { id }`,
          resolve: (parent, args, context, info) =>
            info.mergeInfo.delegateToSchema({
              schema: ratingSchema,
              operation: "query",
              fieldName: "ratingsForBeer",
              args: { beerId: parent.id },
              context, info
            });
        }
      }
    }
  }
}
```

LINK SCHEMAS

Add Resolver

Delegate execution to the other schema ("Rating" in our example)

```
function createLinkedSchema(ratingSchema) {
  return {
    linkedTypeDefs: `extend type Beer {ratings: [Rating!]! }`,
    linkedResolvers: {
      Beer: {
        ratings: {
          fragment: `fragment BeerFragment on Beer { id }`,
          resolve: (parent, args, context, info) =>
            info.mergeInfo.delegateToSchema({
              schema: ratingSchema,
              operation: "query",
              fieldName: "ratingsForBeer",
              args: { beerId: parent.id },
              context, info
            });
        }
      }
    }
  }
}
```

Runs on Rating Schema:

```
query {
  ratingsForBeer(beerId: ...) {
    ...
  }
}
```

MERGE SCHEMAS

Step 4: Merge Schemas together

Create Remote Schemas (as seen before)

```
const beerSchema = await createRemoteSchema("localhost:9010/graphql");
const ratingSchema = await createRemoteSchema("localhost:9020/graphql");

const renamedBeerSchema = renameInSchema(beerSchema, "Beer");
const renamedRatingSchema = renameInSchema(ratingSchema, "Rating");
```

MERGE SCHEMAS

Step 4: Merge Schemas

Create the linked schema (as seen before)

```
const beerSchema = await createRemoteSchema("localhost:9010/graphql");
const ratingSchema = await createRemoteSchema("localhost:9020/graphql");

const renamedBeerSchema = renameInSchema(beerSchema, "Beer");
const renamedRatingSchema = renameInSchema(ratingSchema, "Rating");

const { linkedTypeDefs, linkedResolvers } = createLinkedSchema(renamedRatingSchema);
```

MERGE SCHEMAS

Step 4: Merge Schemas

Create merged schema with remote schema and linked schema

```
import { mergeSchemas } from "graphql-tools";

const beerSchema = await createRemoteSchema("localhost:9010/graphql");
const ratingSchema = await createRemoteSchema("localhost:9020/graphql");

const renamedBeerSchema = renameInSchema(beerSchema, "Beer");
const renamedRatingSchema = renameInSchema(ratingSchema, "Rating");

const { linkedTypeDefs, linkedResolvers } = createLinkedSchema(renamedRatingSchema);

const mergedSchema = mergeSchemas({
  schemas: [
    renamedBeerSchema,
    renamedRatingSchema,
    linkedTypeDefs
  ],
  resolvers: linkedResolvers
});
```

PUBLISH MERGED SCHEMA

Step 5: Publishing

Expose Schema via HTTP endpoint using Express

```
import { graphqlExpress } from "apollo-server-express";
import express from "express";
import bodyParser from "body-parser";

// as seen before
const mergedSchema = mergeSchemas(. . .);

// create Express
const app = express();

// expose our schema
app.use("/graphql",
  bodyParser.json(),
  graphqlExpress({ schema: mergedSchema })
);

// run Express
app.listen(PORT);
```

Thank you!

Slides: https://bit.ly/nordic-coding-graphql

Sample-Code: https://bit.ly/graphql-stitching-example