INFINUM

THE FRONT SIDE OF GRAPHQL

LEO BRDAR

01

GRAPHQL PROS & CONS

GRAPHQL PROS

- no over-fetching or under-fetching
- no N+1 problem
- no multiple roundtrips
- endpoint management
- response format is the same as query
- easy to start with
- great built-in documentation
- typed

GRAPHQL CONS

- JSON representation only
- no automatic scaling & performance optimizations
- no maximum query depth, possible recursions
- hard to rate limit
- hard to implement custom caching mechanism
- error handling (status codes)

02

RELAY

GENERAL IDEA

- JS framework for data-driven React apps with GraphQL
- efficient, decouples client apps
- declarative data fetching

- each component declare only what it needs
- containers / fragments
- root Query / fragments

PACKAGES

- react-relay
- babel-plugin-relay
- graphql
- relay-compiler

```
"scripts": {
  ”relay": "relay-compiler --src ./src --schema ./schema.graphql --extensions js jsx --watch"
}
```

- relay-runtime

RELAY ENVIRONMENT

- defines:
  - configuration
  - cache
  - network layer
- usually a singleton

- batching query requests
- caching
- authentication
- request retrying
- logging

BASIC INITIALIZATION

```
import { Environment, Network, RecordSource, Store } from 'relay-runtime';

function fetchQuery(operation, variables) {
  return fetch('/graphql', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({ query: operation.text, variables }),
  })
    .then(res => res.json());
}

const environment = new Environment({
  network: Network.create(fetchQuery),
  store: new Store(new RecordSource()),
});
```

QUERYRENDERER

- component
- takes query, fetches data and renders it
- props:
  - environment
  - query
  - cacheConfig?
  - variables
  - render ({ error, props, retry }) => React.Node

- fetchQuery – in case you need data outside components

FRAGMENTS

- naming *<FileName>_<propName>*
- colocate components & their data requirements
- de-couples component from the query that renders it
- data masking — all dependecies must be declared!
- *@argumentDefinitions / @arguments*
  - accept / send variables into the fragment
- Fragment container:
  - HOC for components to specify data requirements
  - not fetching directly - must be in QueryRenderer
  - *createFragmentContainer(component, fragmentSpec)*
-

MUTATIONS

```
commitMutation(
  environment,
  config: {
    mutation,
    variables,

    onCompleted? (res, err),
    onError? (err),

    optimisticResponse?,
    optimisticUpdater? (store),
    updater? (store, data),

    configs?  ——> types:  NODE_DELETE, RANGE_ADD, RANGE_DELETE
  }
)
```

03

DEMO

04

RELAY PROS & CONS

RELAY PROS

- encapsulation
- colocation of components with their data dependencies
- declarative
- data masking
- static queries & ahead-of-time code generation
- supports pagination out of the box

- developed and maintained by Facebook

- Relay DevTools Chrome Extension

RELAY CONS

- confusing docs
- a lot of behind-the-scenes “magic“
- hard to start with
- heavily opinionated
- IDs need to be unique across ALL types
- requires custom schema

05

ALTERNATIVES

APOLLO

- GraphQL Client

PROS:

- easier setup
- better docs and overall DX
- can be used as state management
- more customizable
- rapid development

CONS:

- needs to be bundled with parser
- harder to implement pagination
- more manual work
- rapid developement

JSON:API

- specification for building APIs in JSON

PROS:

- pagination is a part of the standard
- JSON pointer in errors

CONS:

- query format
- possibility of not knowing the response format
- no types in specification
- not self documenting
- hard to implement file upload

INFINUM

Any questions?

LEO.BRDAR@INFINUM.CO

Visit infinum.co or find us on social networks:

infinum.co infinumco infinumco infinum