

Now this is GraphQL

Martin Cartledge





Goals

Objective: Build a solution to help Anakin and his pit crew view and modify podracing performance metrics

You will walk away knowing the answers to the following:

- What is GraphQL?
- How is GraphQL different from REST?
- How do I write a schema?
- How do I get and receive data?
- How fast does Anakin's Podracer go?



Setting up

- Project cloned locally
- Open terminal run the following commands:
`npm install` and then `npm start`
- Open code in IDE of your choice
- Navigate to `localhost:4000` in your browser
(won't work just yet)



What is GraphQL?

A query language for your API which empowers the client to have complete control over the data it receives and sends



REST vs GraphQL

REST

REpresentational State Transfer



- Uses constructs that should be familiar to those who have used HTTP
- HTTP status codes
- Language agnostic
- Stateless, all state is managed by the client

Common REST use case

`/podracers/<1>`

```
{
  "podracers": [
    {
      "name": "Anakin Skywalker",
      "location": "tattooine",
      "species": "human",
      "height": "172",
      "mass": "77",
    }
  ]
}
```

`/podracers/<1>/wins`

```
{
  "podrace_wins": [
    "The Boonta Classic",
  ]
}
```

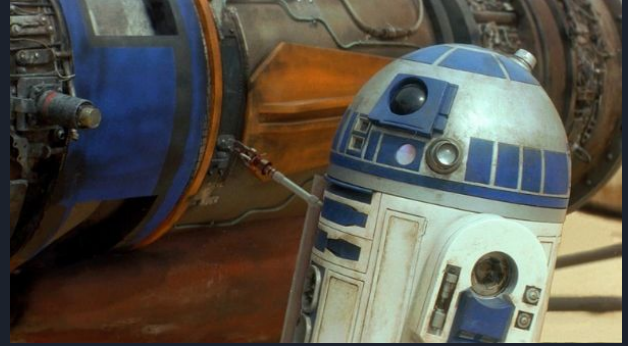
`/podracers/<1>/stats`

```
{
  "podracer_stats": {
    "length": "11",
    "top_speed": "254",
    "average_fuel": "18"
  }
}
```

Code can be found here: [rest.md](#)



GraphQL



- Uses syntax that describes how you ask for you data
- Client specifies what it needs, gets nothing else
- Data is defined by types
- Easy to iterate (update API) because client controls the data it gets

Common GraphQL use case

```
query {  
  getSinglePodracer(id: "2") {  
    name  
    location  
    height  
    mass  
    podrace_wins {  
      title  
    }  
    podracer_stats {  
      length  
      top_speed  
      average_fuel  
    }  
  }  
}
```

```
{  
  "getSinglePodracer": {  
    "name": "Anakin Skywalker",  
    "location": "tattooine",  
    "height": "172",  
    "mass": "77",  
    "podrace_wins": [  
      { "title": "The Boonta Classic" }  
    ],  
    "podracer_stats": {  
      "length": "8",  
      "top_speed": "276",  
      "average_fuel": "29"  
    }  
  }  
}
```

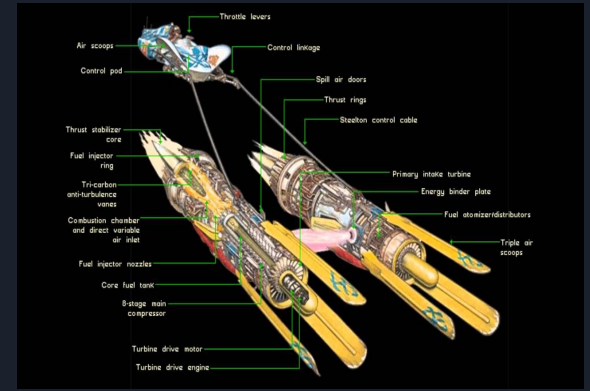
Code can be found here: [graphql.md](#)

Schema

- The blueprint that defines the data type of each object
- Defines the shape of queries for your data

Next, let's create a "Podracer" type!

Code can be found here: `schema.js`





Queries

- How you ask for and receive data
- A string that is sent to a server that is interpreted and that returns JSON to client
- Traverses related objects and fields

Next: let's make a queries to get all Podracers or a specific Podracer!

Code can be found here: `schema.js`



Mutations

- Used to change data (create, update, delete)
- Do not use Query when makes server-side data changes

Next: let's add the ability to create a new Podracer!

Code can be found here: `schema.js`



Resolvers

- Function that returns a value for specified type or a field in *a schema*
- Generates a response for a query
- Contains four arguments:
 - Parent - result returned from resolver on parent field
 - Args - arguments passed to the query
 - Context - a value provided to each resolver
 - Info - a value that holds field-specific info for current query

Next: let's write resolvers for our queries and mutation!
This will allow our queries and mutation to be executed.

Code can be found here: `resolvers.js`

Q & A

