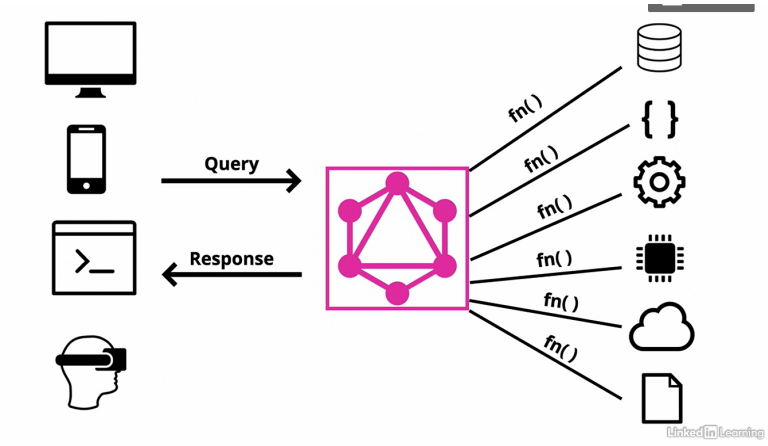
GraphQL

What is GraphQL

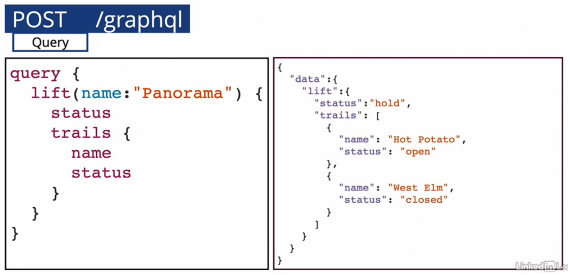
GraphQl is a query language for your API.

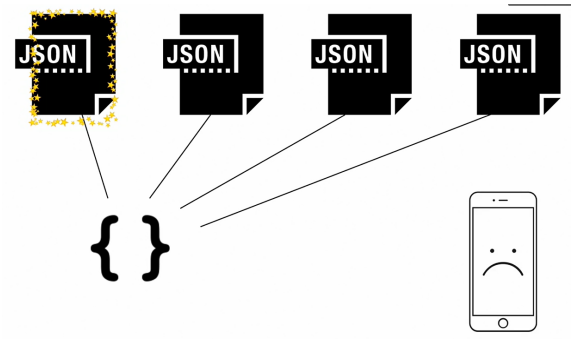
Created by facebook and open soursed in 2015.



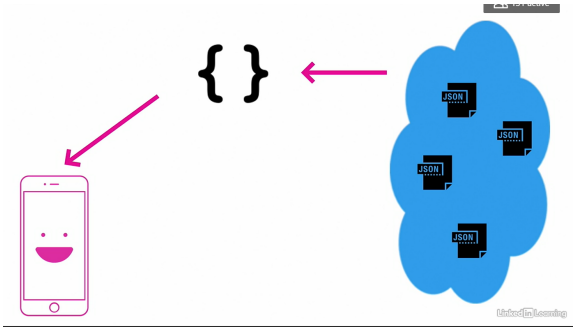
How do we get data with REST

How do we get data with GraphQL



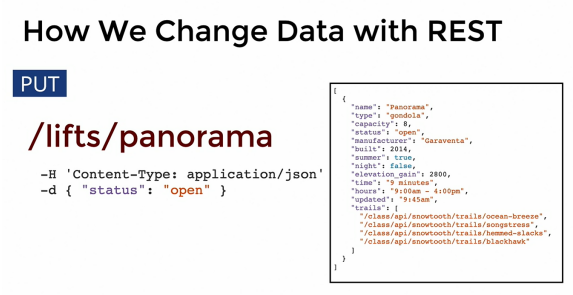


Lots of efforts.

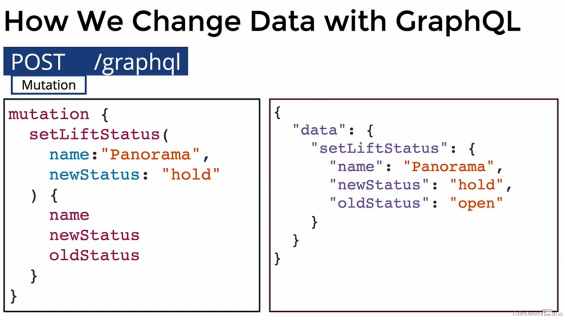


We only get the data which we ask for, no more and no less → which is pretty cool

How we change Data with REST



How we change Data with GraphQL



GraphQL Playground

https://snowtooth.moonhighway.com/

Hit ctrl+space to see all the possible fields that can be queried.

▼ Practice (Using Enum types and arguments)

Query

```
query{
  openLift: liftCount(status: OPEN)
  closedLift: liftCount(status:CLOSED)
  trailCount
  allLifts{
    name,
    capacity,
    id
  }
}
```

Output

```
{
  "data": {
    "openLift": 6,
    "closedLift": 2,
    "trailCount": 71,
    "allLifts": [
      {
        "name": "Astra Express",
        "capacity": 3,
        "id": "astra-express"
      },
      {
        "name": "Jazz Cat",
        "capacity": 2,
        "id": "jazz-cat"
      },
      {
        "name": "Jolly Roger",
        "capacity": 6,
        "id": "jolly-roger"
      },
      {
        "name": "Neptune Rope",
        "capacity": 1,
        "id": "neptune-rope"
      },
      {
        "name": "Panorama",
        "capacity": 8,
        "id": "panorama"
      },
      {
        "name": "Prickly Peak",
        "capacity": 3,
        "id": "prickly-peak"
      },
      {
        "name": "Snowtooth Express",
        "capacity": 6,
        "id": "snowtooth-express"
```

```
      },
      {
        "name": "Summit",
        "capacity": 6,
        "id": "summit"
      },
      {
        "name": "Wally's",
        "capacity": 2,
        "id": "wallys"
      },
      {
        "name": "Western States",
        "capacity": 6,
        "id": "western-states"
      },
      {
        "name": "Whirlybird",
        "capacity": 2,
        "id": "whirlybird"
      }
    ]
  }
}
```

▼ Practice (Adding variables)

Query

```
query($status: LiftStatus){
  liftCount(status: $status)
}
```

Input

```
{
  "status": "CLOSED"
}
```

Output

```
{
  "data": {
    "liftCount": 2
  }
}
```

▼ Practice (Querying Connected types)

Query

```
query{
  allLifts{
    trailAccess{
      name
      status
      accessedByLifts{
        name
      }
    }
  }
}
```

Output

```
{
  "data": {
    "allLifts": [
      {
        "trailAccess": [
          {
            "name": "Blue Bird",
            "status": "OPEN",
            "accessedByLifts": [
              {
                "name": "Astra Express"
              }
            ]
          },
          {
            "name": "Blackhawk",
            "status": "OPEN",
            "accessedByLifts": [
              {
                "name": "Astra Express"
              },
              {
                "name": "Panorama"
              }
            ]
          },
          ......
```

▼ Practice (Creating operation name)

“AllLifts” and “ AllTrails” are called Operation names.

Query

```
query AllLifts{
  allLifts{
    liftName: name
  }
}

query AllTrails{
  allTrails{
    name
    status
  }
}
```

Output for “AllLifts”

```
{
  "data": {
    "allLifts": [
      {
        "liftName": "Astra Express"
      },
      {
        "liftName": "Jazz Cat"
      },
      {
        "liftName": "Jolly Roger"
      },
      ......
    ]
  }
}
```

Output for “AllTrails”

```
{
  "data": {
    "allTrails": [
      {
        "name": "Blue Bird",
        "status": "OPEN"
      },
      {
```

```
        "name": "Blackhawk",
        "status": "OPEN"
      },
      ......
    ]
  }
}
```

▼ Practice (Changing data with mutation)

Mutation is used when you need to change data.

Query

```
mutation{
  setTrailStatus(id:"parachute" status: CLOSED){
    id
    name
    status
  }
}
```

Ouput

```
{
  "data": {
    "setTrailStatus": {
      "id": "parachute",
      "name": "Parachute",
      "status": "CLOSED"
    }
  }
}
```

▼ Practice (Creating GraphQL Fragments)

Query

```
query{
  allLifts{
```

```
    ...LiftDetails
  }
  Lift(id:"summit"){
    ...LiftDetails
  }
}

fragment LiftDetails on Lift{
  id
  name
  status
  capacity
  night
}
```

Output


```
{
  "data": {
    "allLifts": [
      {
        "id": "astra-express",
        "name": "Astra Express",
        "status": "OPEN",
        "capacity": 3,
        "night": false
      },
      {
        "id": "jazz-cat",
        "name": "Jazz Cat",
        "status": "OPEN",
        "capacity": 2,
        "night": false
      },
      {
        "id": "jolly-roger",
        "name": "Jolly Roger",
        "status": "OPEN",
        "capacity": 6,
        "night": true
      },
      {
        "id": "neptune-rope",
        "name": "Neptune Rope",
        "status": "OPEN",
        "capacity": 1,
        "night": false
      },
      {
        "id": "panorama",
        "name": "Panorama",
        "status": "HOLD",
        "capacity": 8,
        "night": false
```

```
      },
      {
        "id": "prickly-peak",
        "name": "Prickly Peak",
        "status": "OPEN",
        "capacity": 3,
        "night": false
      },
      {
        "id": "snowtooth-express",
        "name": "Snowtooth Express",
        "status": "OPEN",
        "capacity": 6,
        "night": false
      },
      {
        "id": "summit",
        "name": "Summit",
        "status": "CLOSED",
        "capacity": 6,
        "night": false
      },
      {
        "id": "wallys",
        "name": "Wally's",
        "status": "HOLD",
        "capacity": 2,
        "night": false
      },
      {
        "id": "western-states",
        "name": "Western States",
        "status": "CLOSED",
        "capacity": 6,
        "night": false
      },
      {
        "id": "whirlybird",
        "name": "Whirlybird",
        "status": "HOLD",
        "capacity": 2,
        "night": false
      }
    ],
    "Lift": {
      "id": "summit",
      "name": "Summit",
      "status": "CLOSED",
      "capacity": 6,
      "night": false
    }
  }
}
```

▼ Practice (Working with subscription)

Subscription will setup a listener. It will listen to any changes as soon as they come in.

It sets up the real time field in your application.

Change data over web socket in real time.

Ex: Implemented in facebook to count the live likes on the posts.

Query

```
subscription{
  liftStatusChange{
    name
    status
  }
}
```

Output

```

```

💡 Queries get data.

💡 Mutations change data.

💡 Subscriptions listen to data changes over a web socket in real time.

GraphQL Playground

https://pet-library.moonhighway.com/

Schema Definition Language (SDL)

GraphQL Scalar Types

- Int
- Float
- String
- Boolean
- ID

```
type Photo{
  id: ID!
  name: String!
  url: String!
  description: String
  rating: Float
  private: Boolean!
}
```

Nullable vs Non-nullable

Non-nullable

```
name:String!
```

Nullable

```
description: String
```

Root Queries

```
type Query{
  totalusers: Int!
}
```

Lists

```
type User{
  postedPhotoes: [Photo!]!
}
```

- photos: [Photo] → Nullable list of nullable values
- photos: [Photo]! → Not-nullable list of nullable values
- photos: [Photo!]! → Not-nullable list of not-nullable values

Setting up a GraphQL server with Apollo Server

Create a folder

```
mkdir ski-day-counter
```

Install dependencies

```
npm init -y
```

Install graphql, apollo-server and nodemon

```
npm install graphql apollo-server nodemon
```

Open in code editor

```
code .
```

Create index.js file which will contain our schema

> 💡 Apollo-server is a NodeJs implementation to a GraphQL server

Create our type defination. So our type defination think of this as just being our schema, and the way that we'll often see this done in a node.js project is we'll use this functio called gql.

```
const typeDefs = gql`

`;
```

gql is the function that comes from Apollo Server package, and its going to take our schema string, and it's going to turn it into and abstract syntax. It's going to turn it into an AST, an abstract syntax tree. (String to object that is little bit easier to parse).

Resolvers are just functions that are just going to return data for the schema.

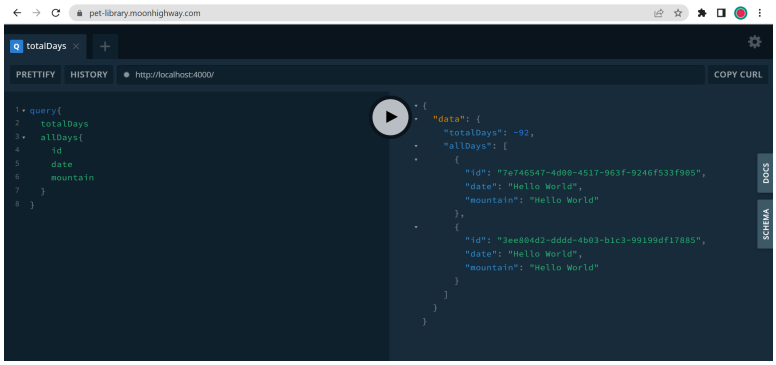
```
const resolvers = {

}
```

Apollo server instance takes two things → typeDefs and resolvers.

```
const server = new ApolloServer({
    typeDefs,
    resolvers
})
```

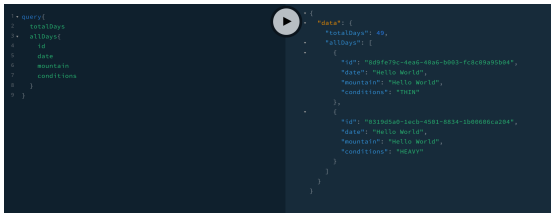
▼ Creating a Custom object

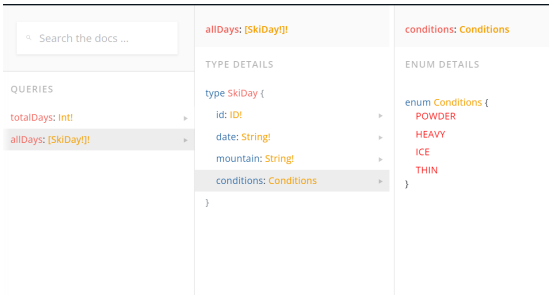
```
const { ApolloServer, gql } = require("apollo-server");

const typeDefs = gql`
    type SkiDay{
        id: ID!
        date: String!
        mountain: String!
    },
  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }
`;

const server = new ApolloServer({
  typeDefs,
  // resolvers
  mocks: true,
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```



▼ Adding an enumaration type

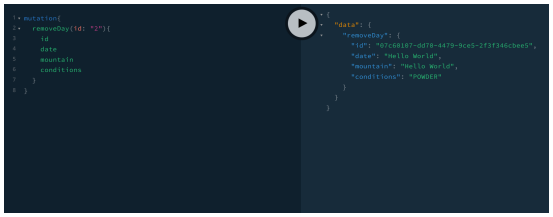
```
const { ApolloServer, gql } = require("apollo-server");

const typeDefs = gql`
  type SkiDay {
    id: ID!
    date: String!
    mountain: String!
    conditions: Conditions
  }

  enum Conditions{
    POWDER
    HEAVY
    ICE
    THIN
  }

  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }
`;

const server = new ApolloServer({
  typeDefs,
  // resolvers
  mocks: true,
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```

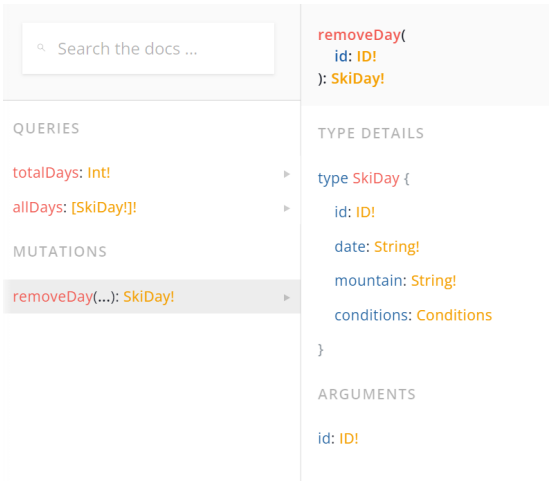




▼ Working with Mutation

```
const { ApolloServer, gql } = require("apollo-server");

const typeDefs = gql`
  type SkiDay {
    id: ID!
    date: String!
    mountain: String!
    conditions: Conditions
  }

  enum Conditions{
    POWDER
    HEAVY
    ICE
    THIN
  }

  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }

  type Mutation {
    removeDay(id: ID!): SkiDay!
  }

`;

const server = new ApolloServer({
  typeDefs,
  // resolvers
  mocks: true,
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```

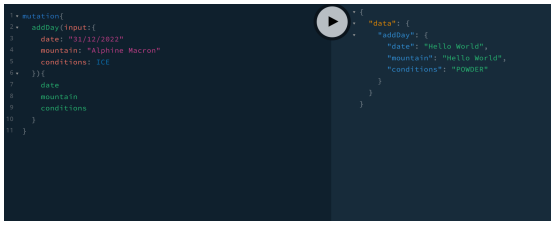


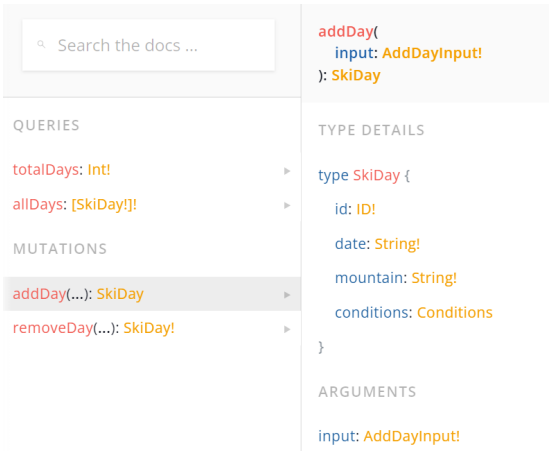


▼ Sending input types to mutations

```
const { ApolloServer, gql } = require("apollo-server");

const typeDefs = gql`
  type SkiDay {
    id: ID!
    date: String!
    mountain: String!
    conditions: Conditions
  }

  enum Conditions {
    POWDER
    HEAVY
    ICE
    THIN
  }

  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }

  input AddDayInput {
    date: String!
    mountain: String!
    conditions: Conditions
  }

  type Mutation {
    addDay(input: AddDayInput!): SkiDay
    removeDay(id: ID!): SkiDay!
  }
`;

const server = new ApolloServer({
```

```
  typeDefs,
  // resolvers
  mocks: true,
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```



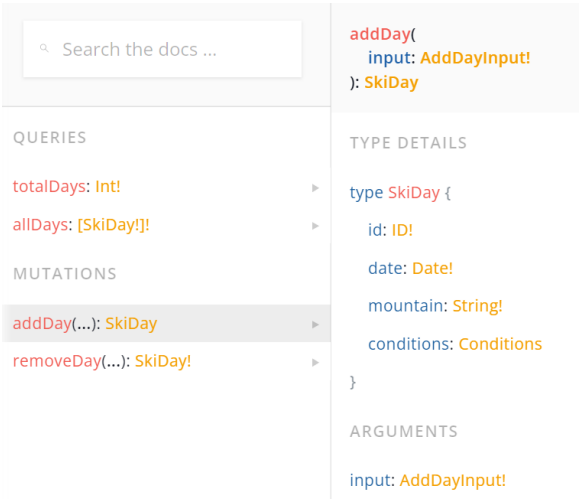


▼ Building a custom scalar

```
const { ApolloServer, gql } = require("apollo-server");

const typeDefs = gql`

  scalar Date

  type SkiDay {
    id: ID!
    date: Date!
    mountain: String!
    conditions: Conditions
  }

  enum Conditions {
    POWDER
    HEAVY
    ICE
    THIN
  }

  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }

  input AddDayInput {
```

```
    date: Date!
    mountain: String!
    conditions: Conditions
  }

  type Mutation {
    addDay(input: AddDayInput!): SkiDay
    removeDay(id: ID!): SkiDay!
  }
`;

const server = new ApolloServer({
  typeDefs,
  // resolvers
  mocks: true,
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```

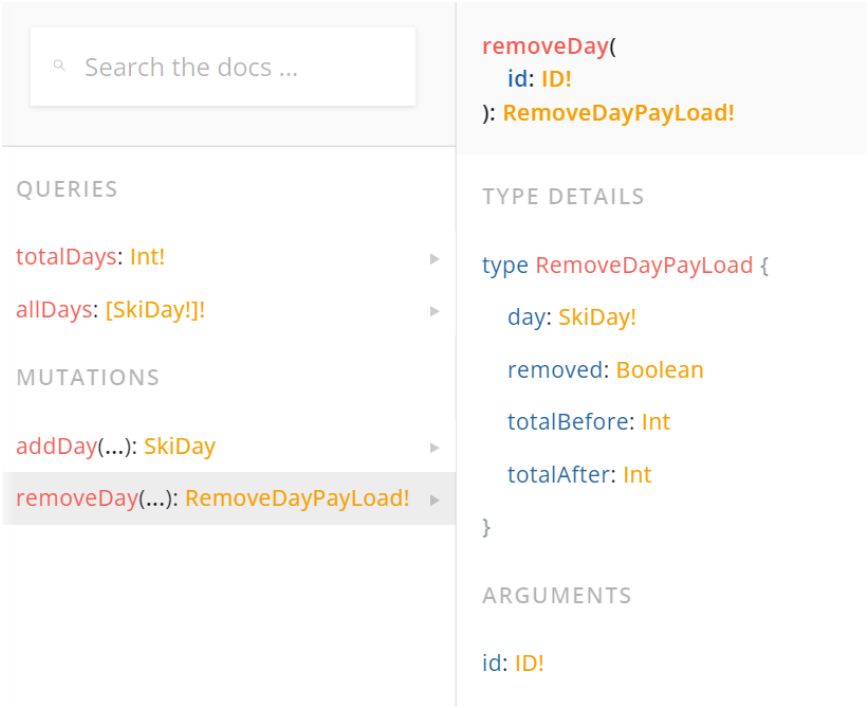
We need to define the scalars i.e., date in the resolver functions. No need to worry while creating the scheme



▼ Returning a custom object

```
const { ApolloServer, gql } = require("apollo-server");

const typeDefs = gql`

  scalar Date

  type SkiDay {
    id: ID!
    date: Date!
    mountain: String!
    conditions: Conditions
```

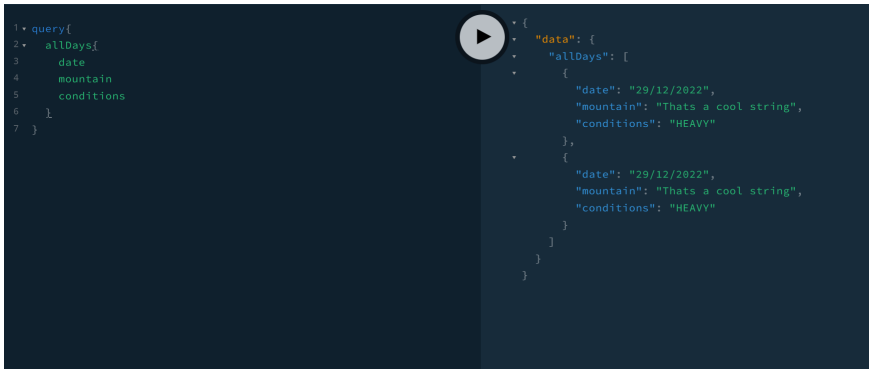
```
  }

  enum Conditions {
    POWDER
    HEAVY
    ICE
    THIN
  }

  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }

  input AddDayInput {
    date: Date!
    mountain: String!
    conditions: Conditions
  }

  type RemoveDayPayLoad{
    day: SkiDay!
    removed: Boolean
    totalBefore: Int
    totalAfter: Int
  }

  type Mutation {
    addDay(input: AddDayInput!): SkiDay
    removeDay(id: ID!): RemoveDayPayLoad!
  }
`;

const server = new ApolloServer({
  typeDefs,
  // resolvers
  mocks: true,
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```



▼ Customizing schema mocks with apollo server

```
const { ApolloServer, gql, MockList } = require("apollo-server");

const typeDefs = gql`
  scalar Date

  type SkiDay {
    id: ID!
    date: Date!
    mountain: String!
    conditions: Conditions
  }

  enum Conditions {
    POWDER
    HEAVY
    ICE
    THIN
  }

  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }

  input AddDayInput {
    date: Date!
    mountain: String!
```

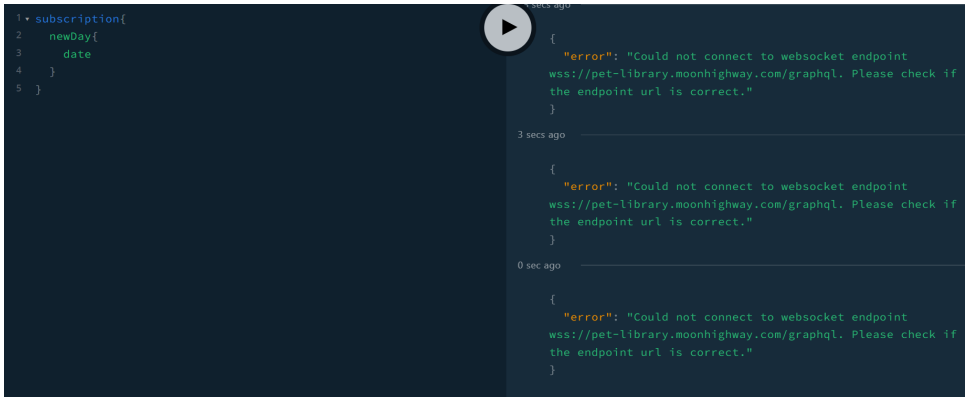
```
    conditions: Conditions
  }

  type RemoveDayPayLoad{
    day: SkiDay!
    removed: Boolean
    totalBefore: Int
    totalAfter: Int
  }

  type Mutation {
    addDay(input: AddDayInput!): SkiDay
    removeDay(id: ID!): RemoveDayPayLoad!
  }
`;

const mocks = {
  Date: () => "29/12/2022",
  String: () => "Thats a cool string",
//   Query: () => ({
//     allDays: () => new MockList(8)
//   })
};

const server = new ApolloServer({
  typeDefs,
  mocks
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```



▼ Creating subscriptions

```
const { ApolloServer, gql, MockList } = require("apollo-server");

const typeDefs = gql`
  scalar Date

  type SkiDay {
    id: ID!
```

```
    date: Date!
    mountain: String!
    conditions: Conditions
  }

  enum Conditions {
    POWDER
    HEAVY
    ICE
    THIN
  }

  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }

  input AddDayInput {
    date: Date!
    mountain: String!
    conditions: Conditions
  }

  type RemoveDayPayLoad{
    day: SkiDay!
    removed: Boolean
    totalBefore: Int
    totalAfter: Int
  }

  type Mutation {
    addDay(input: AddDayInput!): SkiDay
    removeDay(id: ID!): RemoveDayPayLoad!
  }

  type Subscription{
    newDay: SkiDay!
  }
`;

const mocks = {
  Date: () => "29/12/2022",
  String: () => "Thats a cool string",
//   Query: () => ({
//     allDays: () => new MockList(8)
//   })
};

const server = new ApolloServer({
  typeDefs,
  mocks
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```



▼ Wrting schema documentation

```
const { ApolloServer, gql, MockList } = require("apollo-server");

const typeDefs = gql`
  scalar Date

  """
    An object that describes the characteristics of a ski day
  """

  type SkiDay {
    "A ski day's unique identifier"
    id: ID!
    "A date that ski day occurred"
    date: Date!
    "A location where a ski day occurred"
    mountain: String!
    "The shape that the snow was in when this ski day happened"
    conditions: Conditions
  }

  enum Conditions {
    POWDER
    HEAVY
    ICE
    THIN
  }

  type Query {
    totalDays: Int!
    allDays: [SkiDay!]!
  }

  input AddDayInput {
    date: Date!
    mountain: String!
    conditions: Conditions
  }

  type RemoveDayPayLoad{
```

```
    day: SkiDay!
    removed: Boolean
    totalBefore: Int
    totalAfter: Int
  }

  type Mutation {
    addDay(input: AddDayInput!): SkiDay
    removeDay(id: ID!): RemoveDayPayLoad!
  }

  type Subscription{
    newDay: SkiDay!
  }
`;

const mocks = {
  Date: () => "29/12/2022",
  String: () => "Thats a cool string",
//   Query: () => ({
//     allDays: () => new MockList(8)
//   })
};

const server = new ApolloServer({
  typeDefs,
  mocks
});

server.listen().then(({ url }) => console.log(`Server running at ${url}`));
```