GraphQL, REST or RPC? Making the choice!

Rob Allen SymfonyCon, December 2024 APIs can be realised in any style but, which makes the most sense?





• Call a function on a remote server



- Call a function on a remote server
- Common implementations: JSON-RPC, SOAP, gRPC, tRPC



- Call a function on a remote server
- Common implementations: JSON-RPC, SOAP, gRPC, tRPC
- Tends to require a schema (OpenRPC, WSDL, Protocol Buffer)



JSON-RPC

Request:

```
POST / HTTP/1.1
Host: localhost:8545
```

```
"jsonrpc":"2.0",
"id":1,
"method":"createUser",
<u>"params": {"name": "R</u>ob Allen", "email: "rob@akrabat.com"}
```



JSON-RPC

Response:

```
{
    "jsonrpc": "2.0",
    "id":1,
    "result": {"id": 1234}
```





• Operate on a representation of the state of a resource



- Operate on a representation of the state of a resource
- HTTP native



- Operate on a representation of the state of a resource
- HTTP native
- Hypermedia controls



RESTful APIs: Request

POST /users/ Content-Type: application/json Accept: application/json

```
"name": "Rob Allen"
"email": "rob@akrabat.com"
```



{

RESTful APIs: Response

```
HTTP/1.1 201 Created
Content-Type: application/hal+json
ETag: dfb9f2ab35fe4d17bde2fb2b1cee88c1
```

```
"name": "Rob Allen"
"email": "rob@akrabat.com",
"_links": {
    "self": "https://api.example.com/user/1234"
}
```

{



• Retrieve only the data you need on consumer side



- Retrieve only the data you need on consumer side
- Reduce the number of calls to retrieve data with embedded resources



- Retrieve only the data you need on consumer side
- Reduce the number of calls to retrieve data with embedded resources
- Self-describing, typed schema



Queries

```
query {
  author(name: "Anne McCaffrey") {
    id, name
    books(first: 5) {
      totalCount
      edges {
        node {
          id, title
```

Queries

```
query {
  author(name: "Anne McCaffrey") {
    id, name
    books(first: 5) {
      totalCount
      edges {
        node {
         id, title
```





Queries

```
query {
    id, name
    books(first: 5) {
      totalCount
      edges {
        node {
          id, title
```

Queries: Result

```
"data": {
  "author": {
    <u>"id":</u> "MXxBdXRob3J8ZjA",
    "name": "Anne McCaffrey",
    "books": {
      "totalCount": 6,
      "edges":
          "node": {
            "id": "MXxCb29rfGYwNzU",
            "title": "Dragonflight"
```





Queries: Result

```
"data": {
  "author": {
    "id": "MXxBdXRob3J8ZjA",
    "name": "Anne McCaffrey",
    "books": {
      "totalCount": 6,
      "edges": [
          "node": {
            "id": "MXxCb29rfGYwNzU",
            "title": "Dragonflight"
```



Queries: Result

```
"data": {
 "author": {
   "id": "MXxBdXRob3J8ZjA",
    "books": {
      "totalCount": 6,
     "edges":
          "node": {
            "id": "MXxCb29rfGYwNzU",
            "title": "Dragonflight"
```





Which to pick?

Lamborghini or Ferrari?

Q

1



Lamborghini or Truck?



Considerations

- What is it to be used for?
- Response customisation requirements
- HTTP interoperability requirements



What is it to be used for?

- Do you control both server and client?
- How many users are expected?
- What is the skill level of your integrators?



Response customisation

- GraphQL is a query-first language
- REST tends towards less customisation
- With RPC you get what you're given!



Response customisation

- GraphQL is a query-first language
- REST tends towards less customisation
- With RPC you get what you're given!

(Your data layer's ability to efficiently retrieve the data is still key!)



Performance

- REST and RPC puts server performance first
- GraphQL puts client performance first





- RPC, REST and GraphQL can all cache in application layer
- REST can additionally cache at HTTP layer

```
RPC:
   POST /api
   {
    "method": "getAvatar",
    "userId": "1234"
   }
   {
    "result": "(base64 data)"
```



```
RPC:
 POST /api
    "method": "getAvatar",
    "userId": "1234"
    "result": "(base64 data)"
```

```
GraphQL:
  query {
    avatar(userId: "1234")
    "data": {
      "avatar": "(base64 data)"
      "format": "image/jpeg"
```





REST:

GET /user/1234/avatar Accept: application/json

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
____data": "(base64 data)"
```



REST:

GET /user/1234/avatar Accept: application/json

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
"data": "(base64 data)"
```

REST:

GET /user/1234/avatar Accept: image/jpeg

HTTP/1.1 200 OK Content-Type: image/jpeg

<jpg image data>



Errors

- RPC: Returned payload contains an error object of some form
- REST: HTTP semantics; status code
- GraphQL: Top level error object for Request errors and Field errors



REST Errors

HTTP/1.1 503 Service Unavailable Content-Type: application/problem+json Content-Language: en

{

"status": 503, "type": "https://example.com/service-unavailable", "title": "Could not authorise user.", "detail": "Auth service is down for maintenance.", "instance": "https://example.com/maintenance/2023-05-12", "error_code": "AUTHSERVICE_UNAVAILABLE"



GraphQL Errors

```
"errors":
    "message": "Name for character with ID 7 could not be fetched.",
    "path": ["friends", 1, "name"]
"data": {
    "friends": [
      { "id": "3", "name": "F'lar", "species": "human"},
      { "id": "7", "name": null, "species": "dragon" },
      { "id": "9", "name": "Mnementh", "species": "dragon" },
```

Versioning

• RPC, GraphQL and REST can all version via evolution as easily as each other



Versioning

- RPC, GraphQL and REST can all version via evolution as easily as each other
- GraphQL is very good for deprecation of specific fields



Design considerations

It's always hard!



Design considerations

It's always hard!



It's your choice

If you suck at providing a REST API, you will suck at providing a GraphQL API

Arnaud Lauret, API Handyman





Thank you!

Rob Allen 🛛 💥 @akrabat 🛛 @rob@akrabat.com

Photo credits

- Choose Pill: https://www.flickr.com/photos/eclib/4905907267
- Lamborghini & Ferrari: https://akrab.at/3w0yFmg
- Lamborghini & Truck: https://akrab.at/3F4kAZk
- '50s Computer: https://www.flickr.com/photos/9479603@N02/49755349401
- Blackboard: https://www.flickr.com/photos/bryanalexander/17182506391
- Crash Test: https://www.flickr.com/photos/astrablog/4133302216