Web applications with GraphQL & React

@rubydwarf

Hi!

Marion Schleifer
@rubydwarf

hasura / graphql-engine

Watch 215 | Star 12,232 | Fork 948

Code | Issues 524 | Pull requests 85 | Projects 0 | Wiki | Security | Insights

Blazing fast, instant realtime GraphQL APIs on Postgres with fine grained access control, also trigger webhooks on database events.
https://hasura.io

graphql | graphql-server | postgres | hasura | access-control | automatic-api

1,206 commits | **6** branches | **52** releases | **172** contributors | View license

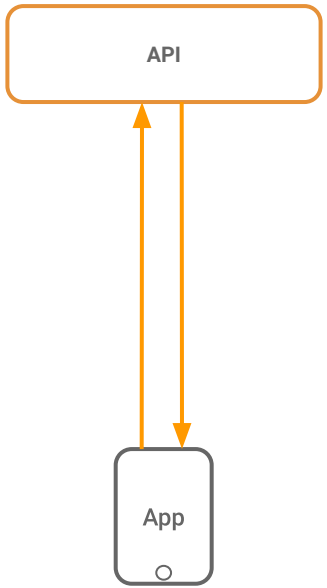

47
YAMAHA
ACERBIS

Agenda

- Intro to GraphQL
- Build a GraphQL API: demo
- GraphQL & React
- Build a real-time app: demo

#1
An API call

An API call (before)



HTTP request

```
GET /api/user?id=1
```

HTTP response

```
{
  "id": 1,
  "name": "Elmo"
}
```

```
GET
/api/address?user_id=1
```

```
{
  "street": "Sesame street",
  "city": "New York City"
}
```

An API call (before)

Yuck! 2 API calls.

It can only get worse with more API calls.

An API call (before)

Let’s talk to our API developer to help us out.

With one API call.

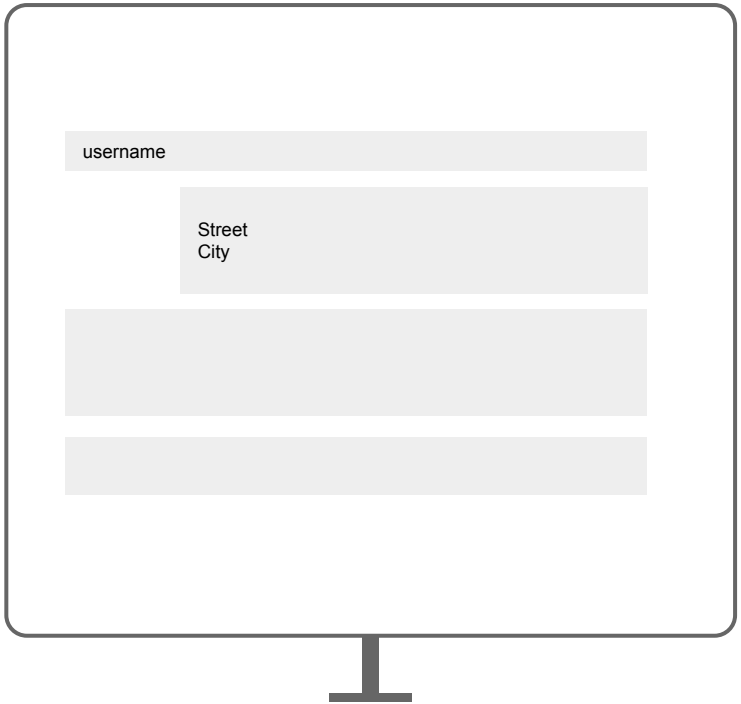
HTTP request

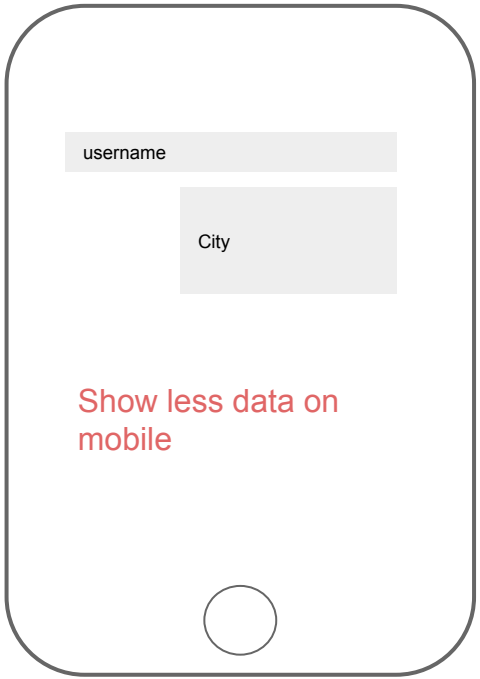
```
GET /api/userinfo?id=1
```

HTTP response

```
{
  "id": 1,
  "name": "Elmo"
  "address": {
     "street": "Sesame street",
     "city": "New York City"

  }
}
```

An API call (before)





An API call (before)

Let's talk to our API developer to help us out. *Again*.

With one API call that *takes params*

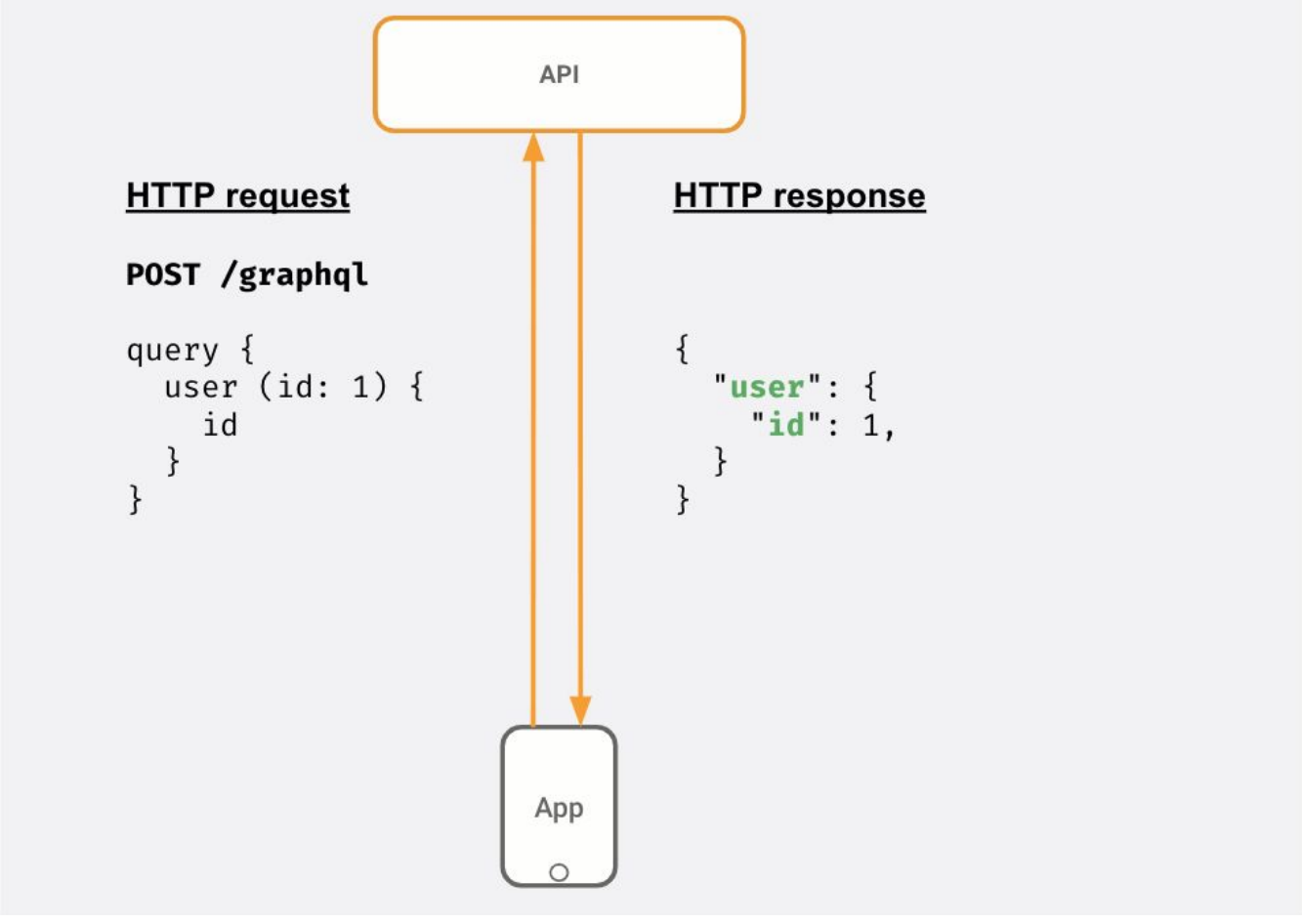
HTTP request

```
GET /api/userinfo?id=1&fields=id,name,address,city
```

HTTP response

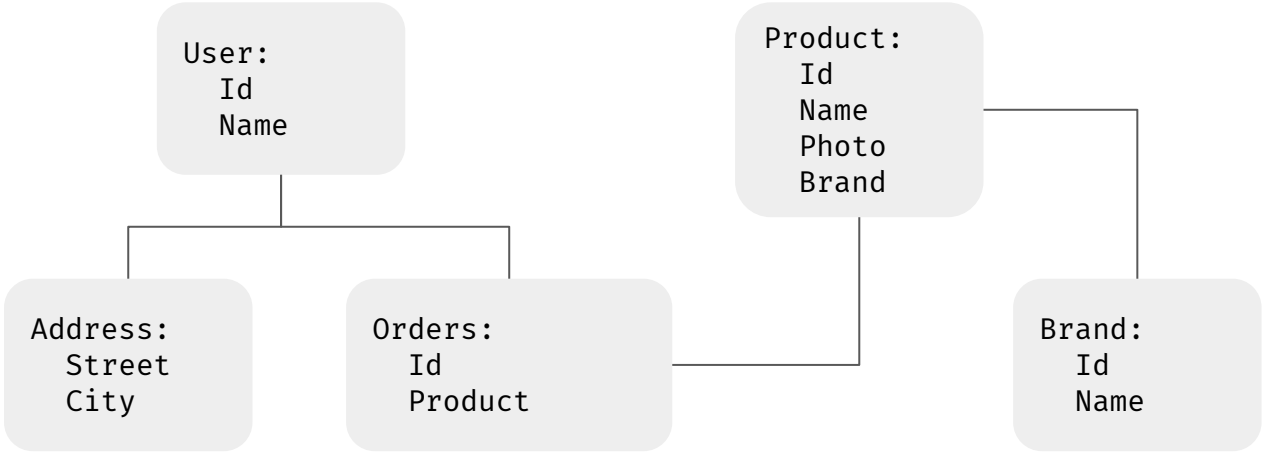
```
{
  "id": 1,
  "name": "Elmo"
  "address": {
     "city": "New York City"
  }
}
```

An API call (after)



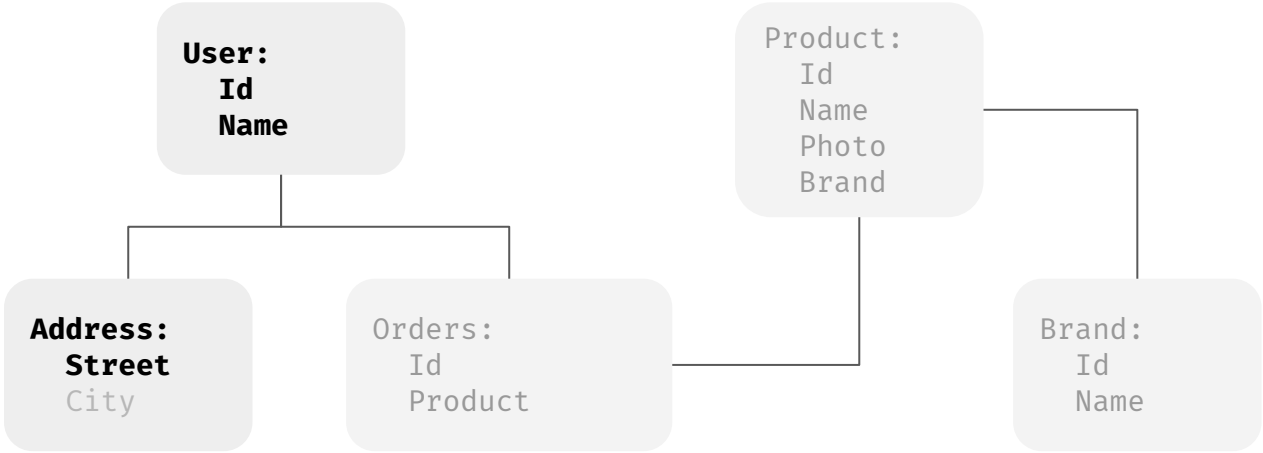
Key insights #1

Your API models are “graph” like.



Key insights #2

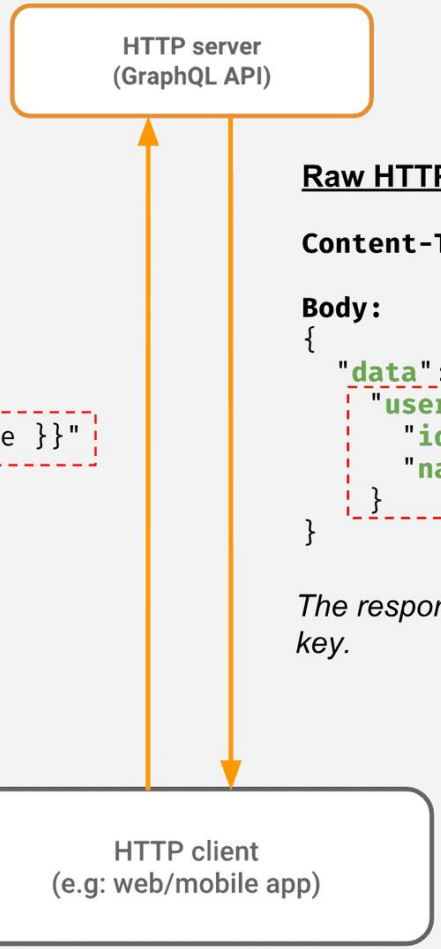
You want to control the data you get



A GraphQL query

```
query {
  user (id: 1) {
    id
    name
    address {
      street
    }
  }
}
```

GraphQL underneath



Raw HTTP request

```
Method:       POST
URL:          https://api.com/graphql
Content-Type: application/json

Body:
{
  "query": "query { user (id: 1) { id name }}"
}
```

The GraphQL query is sent as a string inside a JSON object.

Raw HTTP response

```
Content-Type: application/json

Body:
{
  "data": {
    "user": {
      "id": 1,
      "name": "Elmo"
    }
}
```

*The response object is inside the **`data`** key.*

#2
“Write” APIs

“Writing” to your API (before)

- POST
- PUT
- DELETE
- PATCH

HTTP request

```
POST /api/todo

{
  "todo": "Grok GraphQL"
}
```

HTTP response

```
200

{
  "id": 987
}
```

“Writing” to your API (after)

HTTP request

POST /graphql

```
mutation {
  addTodo(todo: $newTodo) {
    id
  }
}
```

```
{
  "newTodo": {
    "todo": "Grok GraphQL"
  }
}
```

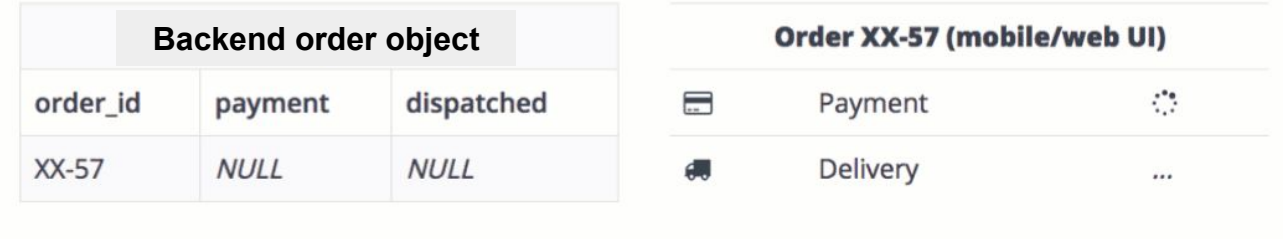
Query variable

HTTP response

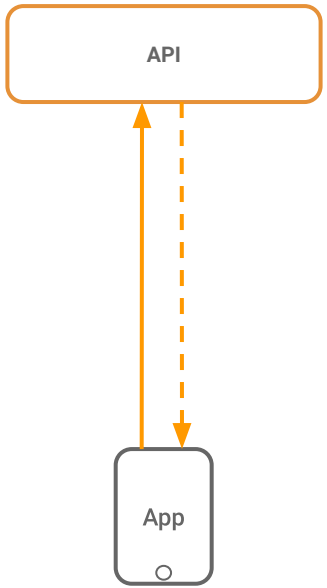
200

```
{
  "id": 987
}
```

#3
“Realtime” APIs



“Realtime” APIs (before)



Option 1: Polling

Client makes repeated requests every X seconds to refetch data.

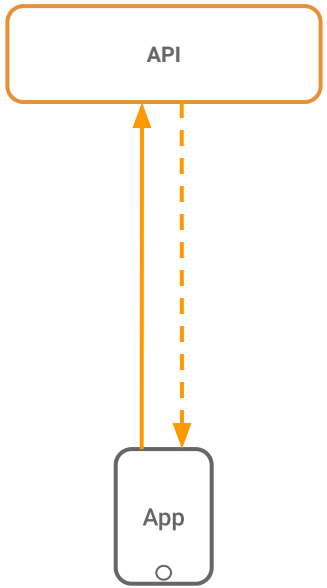
#yuck

Option 2: Websockets

Server pushes data to the client over websockets.

#nightmare

“Realtime” APIs (after)



HTTP request

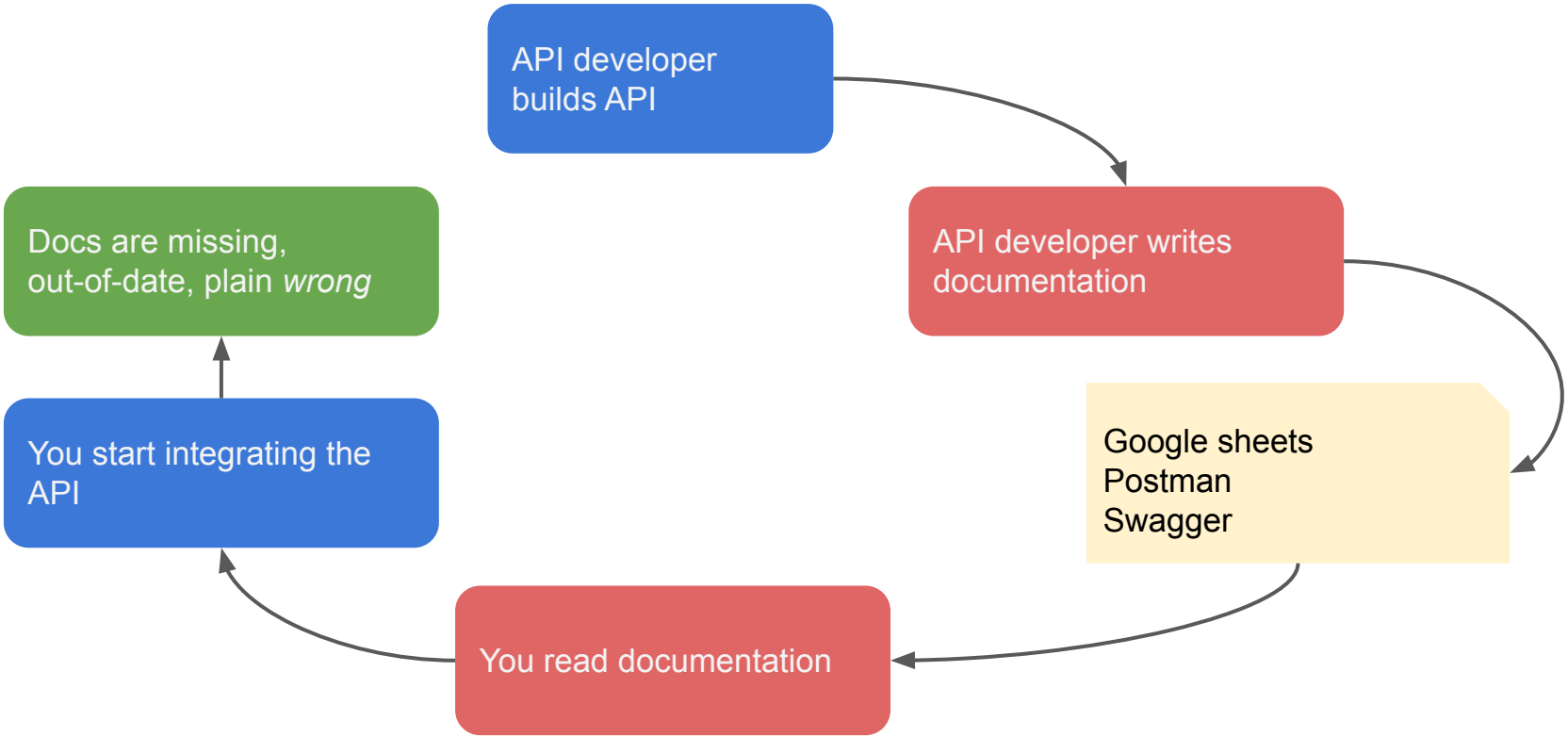
```
ws://myapi.com/graphql

subscription {
  order(id: "XX-57") {
    paid
    dispatched
  }
}
```

HTTP response

```
{
  "paid": true,
  "dispatched": false,
}
```

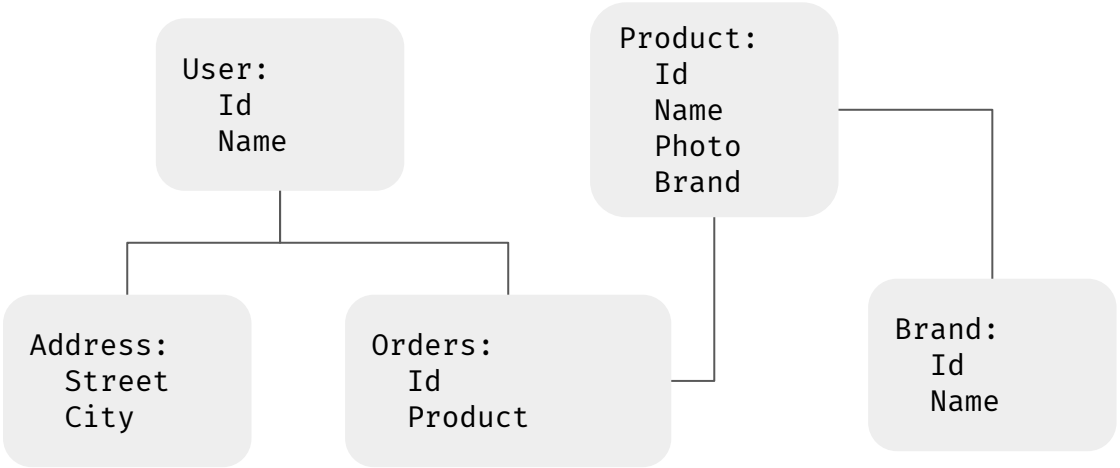
#4
Sharing/documenting APIs

Sharing/documenting APIs (before)



Sharing/documenting APIs (after)

GraphQL schema: The type-system of your API




```
type User {
  id: Int
  name: String
  address: Address
}
```

```
type Address {
  id: Int
  street: String
  city: String
}
```

Introspection API

Make a GraphQL query to *fetch the type information!*

```
{
  __type(name: "todos") {
    name
    fields {
      name
    }
  }
}
```

todos
id
created_at
is_completed
text
user
...

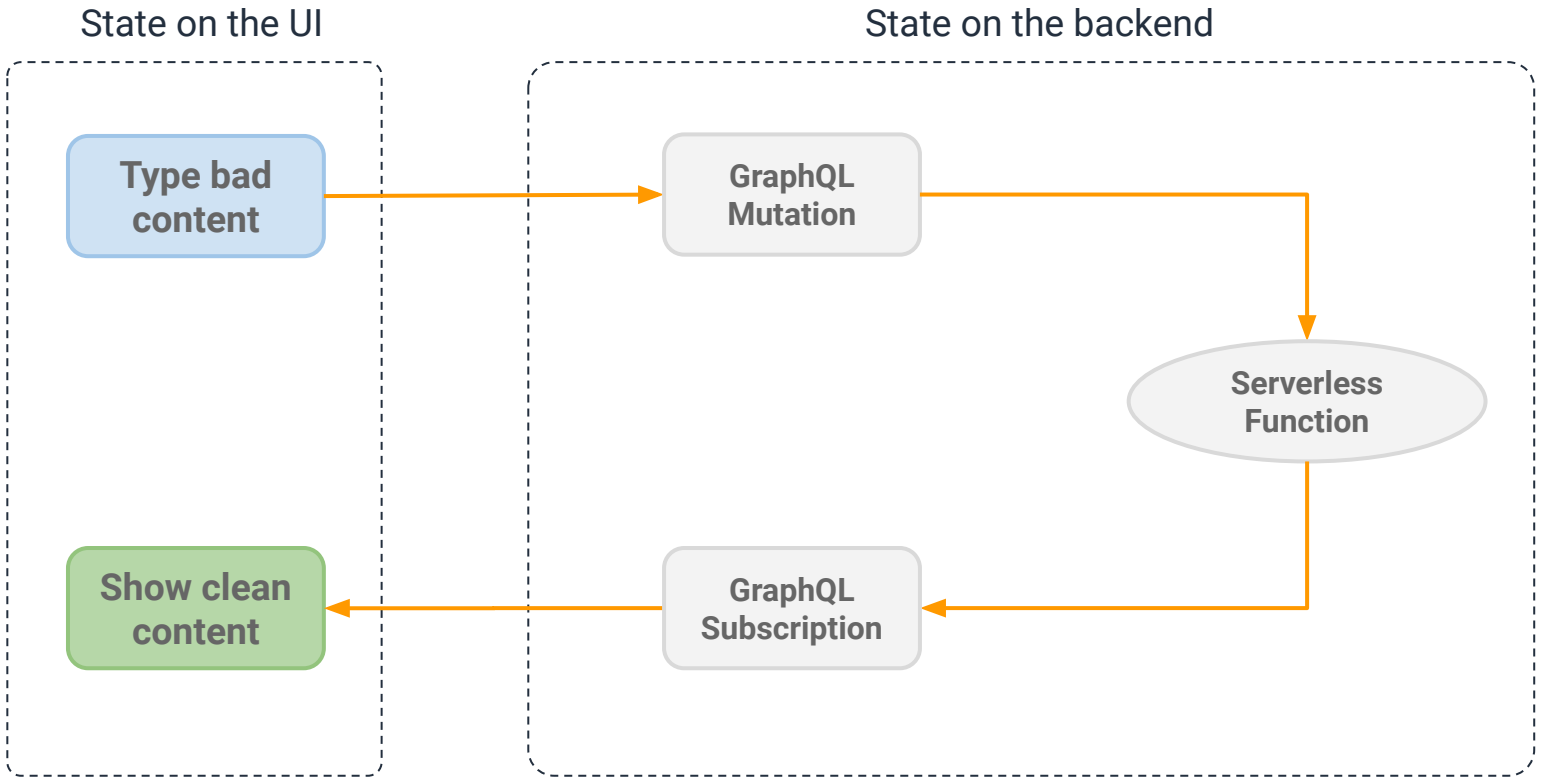
Challenges with adding GraphQL

Challenges

- N + 1 queries to your database / backend services
- Access control and authorization
- GraphQL is ideal for a monolith; Patterns for cloud-native? (microservices, serverless)
  - Microservices
  - Event-driven

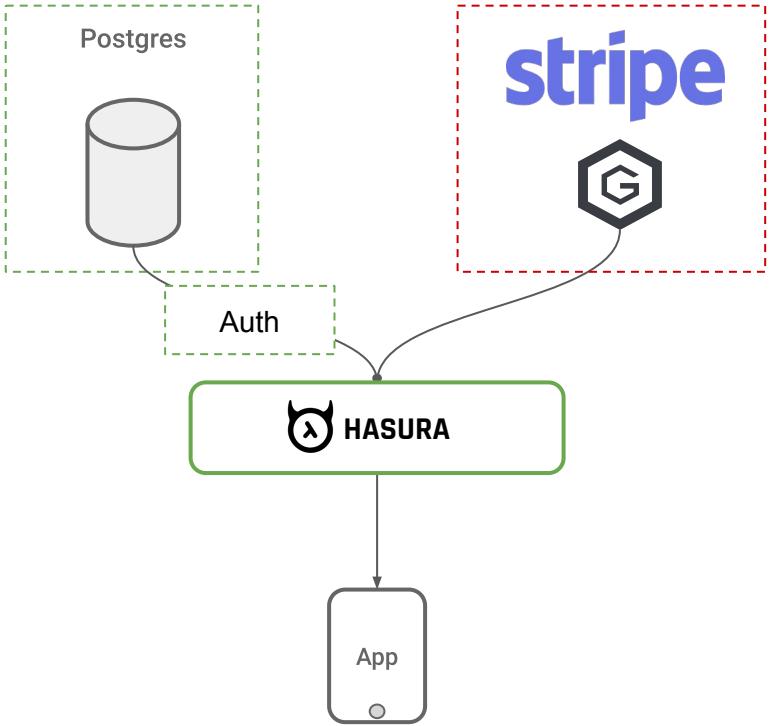

DEMO GODS
PLEASE LET THIS DEMO WORK

Demo app: sanitize text

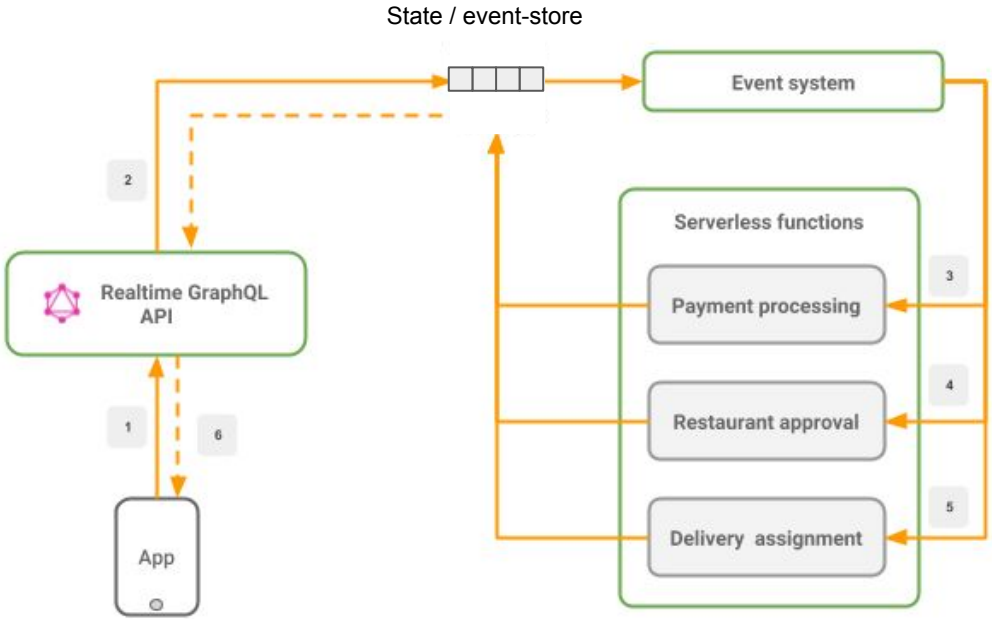

State on the UI
State on the backend
Type bad content
GraphQL Mutation
Serverless Function
Show clean content
GraphQL Subscription

Remote Joins

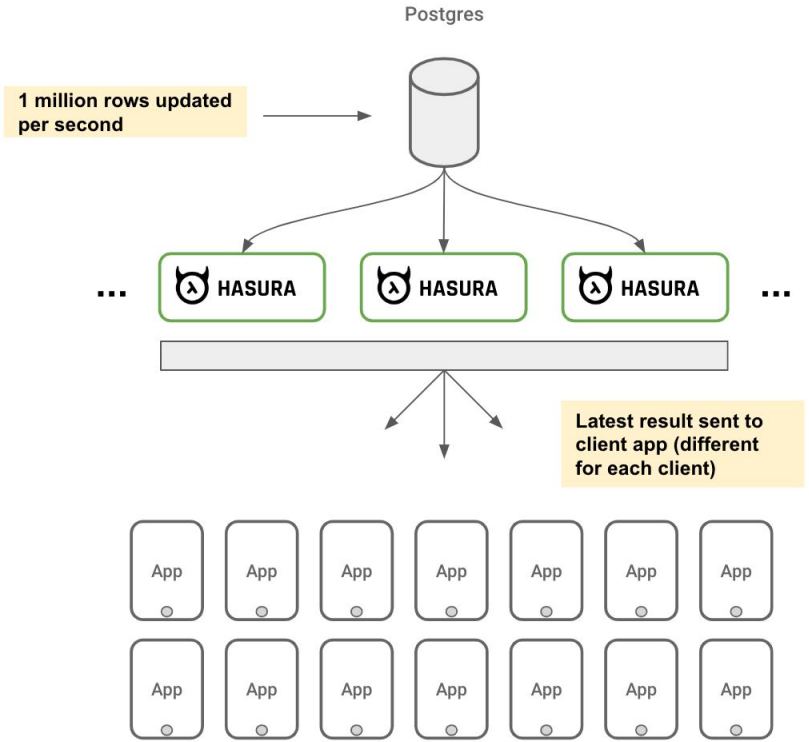
```
query {
  customer {
    id
    email
    stripe {
      account_balance
    }
  }
}
```



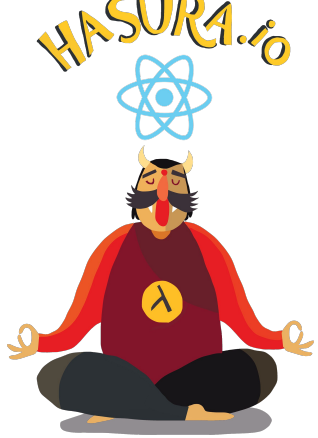
“Flux” style one way data-flow



Scalable, reliable subscriptions



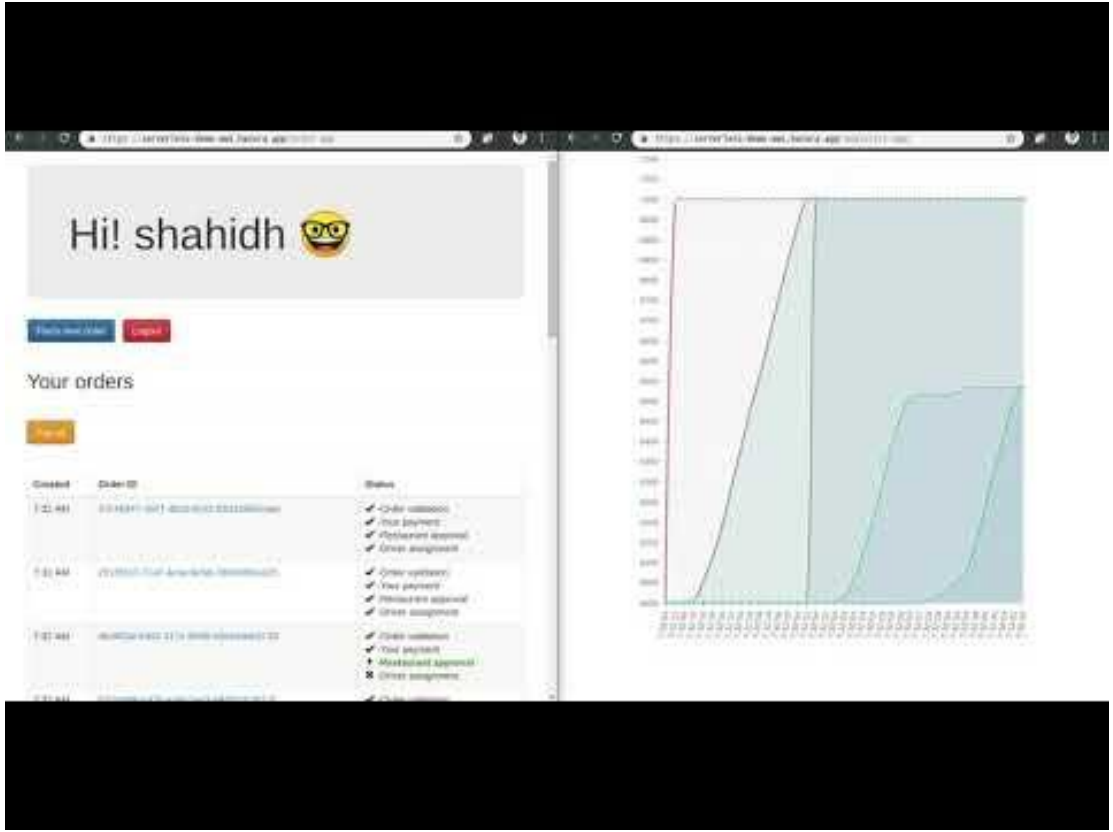
Get started with GraphQL now



Thank you 🙏

Marion Schleifer
@rubydwarf

bit.ly/3factor-app


Hi! shahidh
Your orders

Notes

- Intro to GraphQL
- Intro to Hasura:
  - Auto-generates GraphQL API, adds access control
  - Solves hard challenges with GraphQL
  - Demo: chinook:
    - https://gist.github.com/coco98/fe587d6aa027d24a4dc97d970cb8396b
    - psql postgres://ayrwepftryaufp:7421b064502ab709c57f40401511c6927c024e5fe2abaeb95d9a90954455545e@ec2-34-196-180-38.compute-1.amazonaws.com:5432/d3kem8qcsnbj9u < chinook_pg_serial_pk_proper_naming.sql
    - Queries
    - Subscriptions
  - Event driven business logic
    - Bad words demo
  - Food ordering demo