Prisma / GraphQL

Optimiser son API avec GraphQL (et Prisma)

Robin

Diplômé ESGI IW session 2019
Développeur FS free-lance

thorelrobin@yahoo.fr

Table des matières

GraphQL, c’est quoi?

01

“GraphQL est un langage de requêtes pour API ainsi qu’un environnement pour exécuter des requêtes”.

—DEFINITION OFFICIELLE

GraphQL

Base de données ✗

Framework ✗

Exclusif au langage JavaScript ✗

Langage de requêtage de base de données ✓

Langages de requête (1)

firstName	lastName	dateOfBirth
Durant	Albert	1958/09/10
Dupont	Alphonse	1975/01/23
Dupont	Alice	1963/06/07

Langages de requête (2)

SQL

```
SELECT firstName, dateOfBirth
FROM employees
WHERE lastName = "Dupont"
```

GraphQL

```
query {
 employees(
  where: {
    lastName: "Dupont"
    }
  ) {
    fistName,
    dateOfBirth
  }
}
```

XQuery

```
//FLWOR Syntax

for $b in document ("exemple.xml")
where $b/lastName = "Dupont"
return
  <dupont>{
    $b/firstName,
    $b/dateOfBirth
  }</dupont>
```

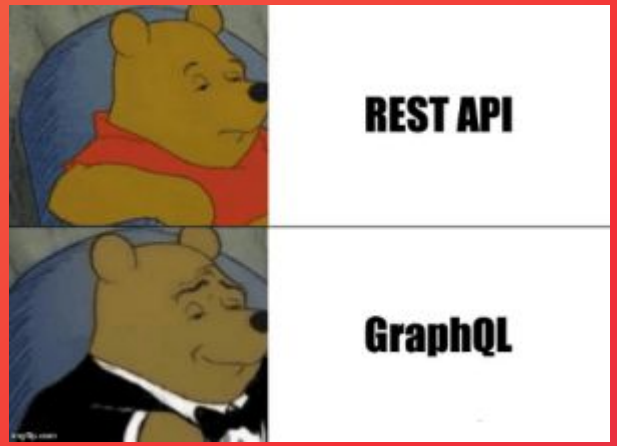
Langages de requête (3)

lastName	dateOfBirth
Alphonse	1975/01/23
Alice	1963/06/07

```
{
  "data": {
    "employees": [
      {
        "firstName": "Alphonse",
        "dateOfBirth":  {
          "day": 23,
          "month": 01,
          "year": 1975,
          "hour": 0,
          "minute": 0,
          "second": 0,
          "formatted": "1975-01-23T00:00:00Z"
        }
      },
      {
        "firstName": "Alice",
        "dateOfBirth":  {
          "day": 07,
          "month": 06,
          "year": 1963,
          "hour": 0,
          "minute": 0,
          "second": 0,
          "formatted": "1963-06-07T00:00:00Z"
        }
      }
    ]
  }
}
```

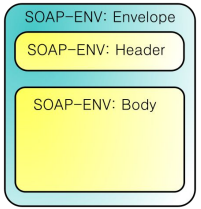
```
<dupont>
  <prenom>Alphonse</prenom>
  <date_naissance>23/12/1975</date_naissance>
</dupont>
<dupont>
  <prenom>Isabelle</prenom>
  <date_naissance>12/03/1967</date_naissance>
</dupont>
```

REST vs GraphQL



02

Les différents types d'API

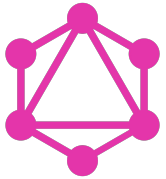


SOAP

Protocole d'échange d'information structurée dans l'implémentation de services web bâti sur XML

REST

Interroger un serveur tiers en utilisant les verbes HTTP pour spécifier l'action et accéder à la ressource



GraphQL

La requête client définit une structure de données, et le serveur suit cette structure pour retourner la réponse

GraphQL vs REST

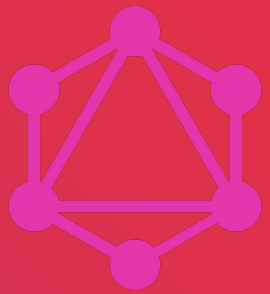
Bonnes idées dans REST: accès structuré aux ressources & stateless server

REST a des spécifications strictes: ce concept a mal été interprété et sauvagement utilisé

L'exigences des évolution rapide côté client ne font pas bon ménage avec la nature statique de REST

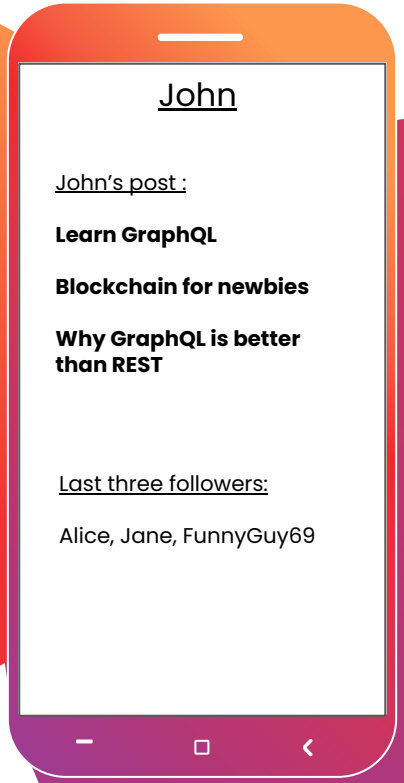
Alternative efficace a REST

- L’utilisation accrue des mobiles a créé un besoin de chargement de données plus efficace
- Variété de différents frameworks frontend
- Vitesse de développement rapide



GraphQL a été développé pour faire face **au besoin de plus de flexibilité et d'efficacité** dans la communication client <-> server

Exemple Blog



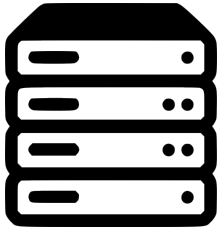
Exemple Blog - REST

3 API Endpoints

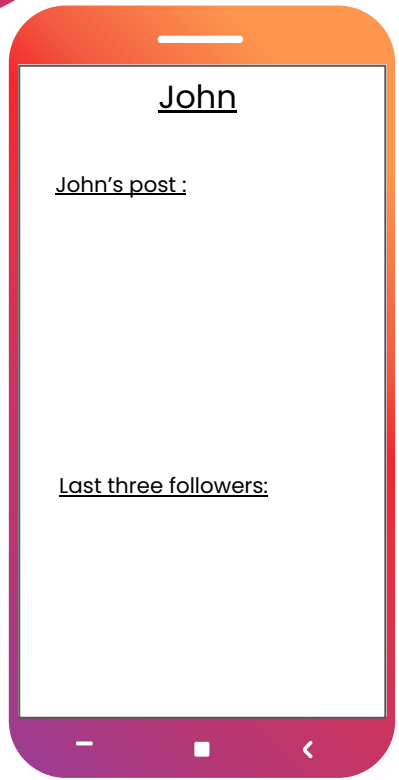
/users/{id}

/users/{id}/posts

/users/{id}/followers



Exemple Blog - REST
1 Fetch data user



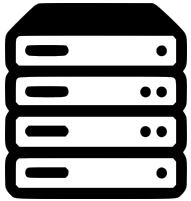
HTTP GET

```
{
    "user": {
        "id": "gj34cm50mpsw",
        "name": "John",
        "address": { ... },
        "birthday": "June 15, 1998"
    }
}
```

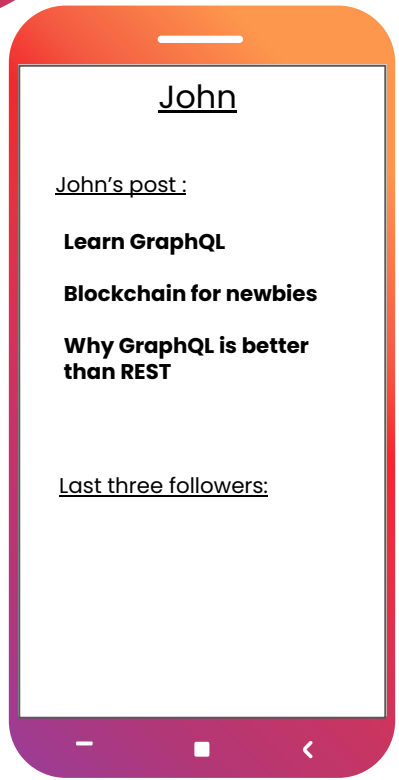
/users/{id}

/users/{id}/posts

/users/{id}/followers



Exemple Blog - REST
2 Fetch data posts



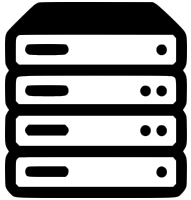
HTTP GET


```
{
    "posts": [{
        "id": "naowp45mn1",
        "title": "Learn GraphQL",
        "content": "Lorem ipsum...",
        "comments": [ ... ],
    }, {
        "id": "zoi15zlx1ckq",
        "title": "Blockchain for...",
        "content": "Lorem ipsum...",
        "comments": [ ... ],
    }, {
        "id": "aq60hzcnqyw9p",
        "title": "Why GraphQL...",
        "content": "Lorem ipsum...",
        "comments": [ ... ]
```

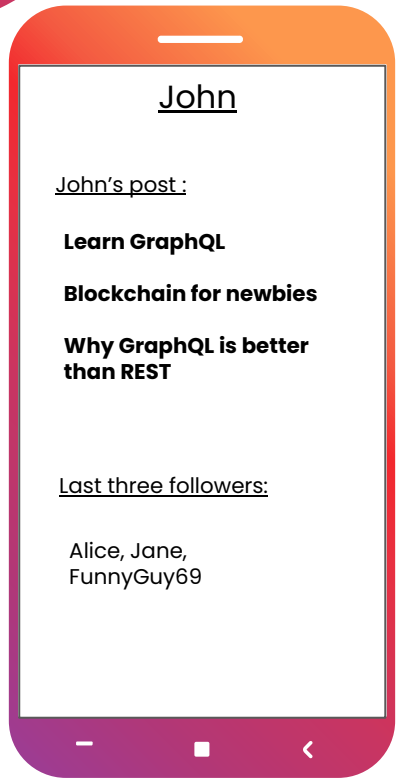

/users/{id}

/users/{id}/posts

/users/{id}/followers



Exemple Blog - REST
3 Fetch data followers



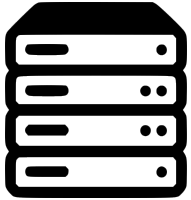
HTTP GET


```
{
    "followers": [{
        "id": "ajk3km5sly4sw",
        "name": "Alice",
        "address": { ... },
        "birthday": "June 15, 1990",
    },{
        "id": "cksi90qnxj1egk",
        "name": "Jane",
        "address": { ... },
        "birthday": "October 7, 1986",
    },{
        "id": "usnh53fhnlp24",
        "name": "FunnyGuy69",
        "address": { ... },
        "birthday": "December 31, 2002",
    },
    ...
```


/users/{id}

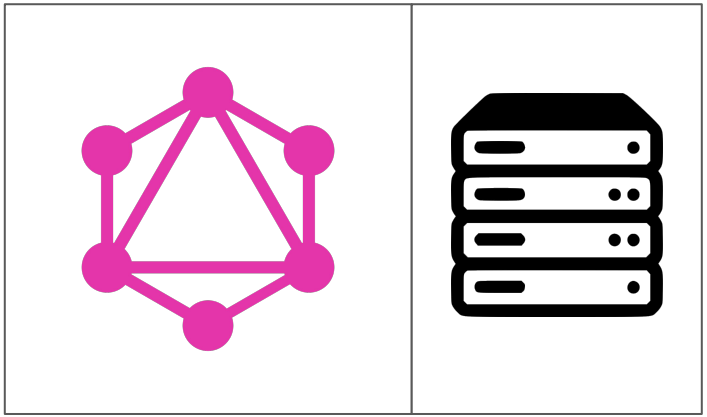
/users/{id}/posts

/users/{id}/followers

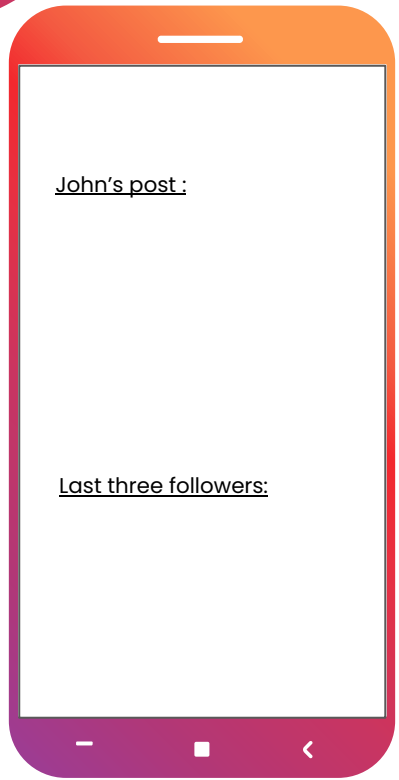


Exemple Blog - GraphQL

1 API Endpoints

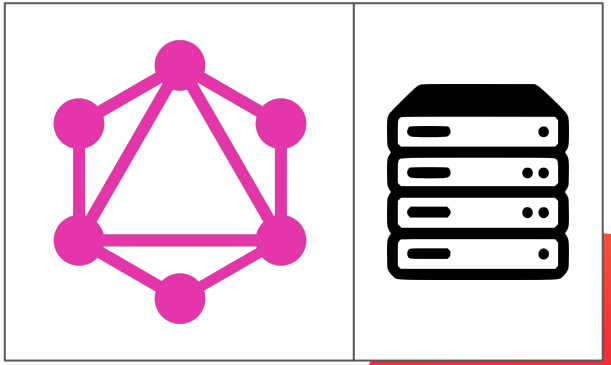


Exemple Blog - REST
1 Fetch all data needed with unique query

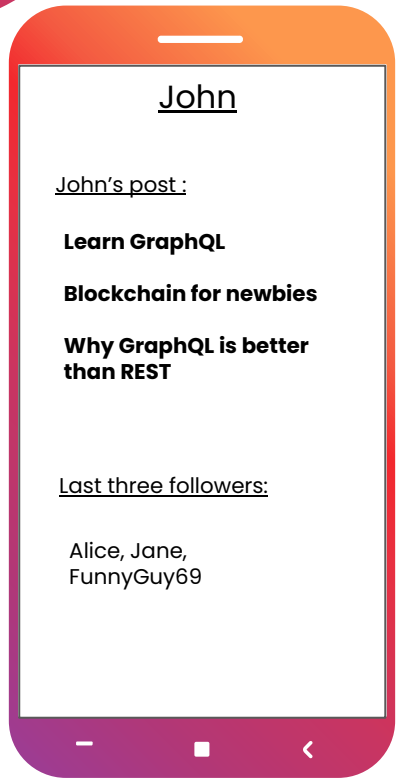


HTTP POST

```
query {
    User(id: "gj34cm50mpsw") {
        name
        posts {
            title
        }
        followers(last: 3) {
            name
        }
    }
}
```

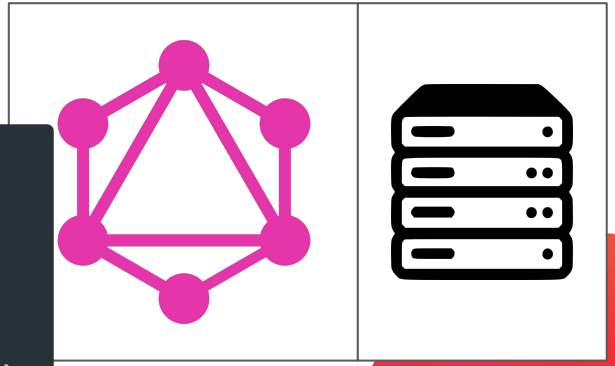


Exemple Blog - REST
1 Fetch all data needed with unique query



HTTP POST

```
{
    "data": {
        "User": {
            "name": "John",
            "posts": [
                { title: "Learn GraphQL" },
                { title: "Blockchain for newbies" },
                { title: "Why GraphQL is better than REST" },
            ],
            "followers": [
                { name: "Alice" },
                { name: "Jane" },
                { name: "FunnyGuy69" },
            ]
        }
    }
}
```



Avantages

Finit le **overfetching** : télécharger plus de données que nécessaire

Plus de **underfetching** : un endpoint ne retourne pas assez de données nécessaire ; besoin d'envoyer de multiples requêtes

Plus besoin d'adapter les endpoints quand les **spécifications du produit et le design change**

Cycles de **feedback et itération produit plus rapides**

03

GraphQL

En détails

Schema Definition Language (SDL)

Definition de types simple

```
type Person {
  name: String!
  age: Int!
}
```

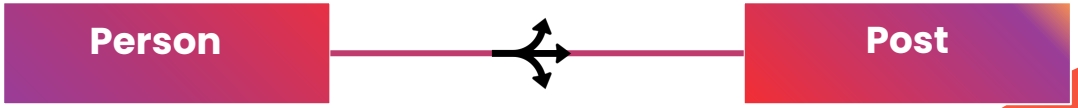
```
type Post {
  title: String!
}
```

Schema Definition Language (SDL)

Ajout de relation

```
type Person {
  name: String!
  age: Int!
  posts: [Post!]!
}
```

```
type Post {
  title: String!
  author: Person!
}
```



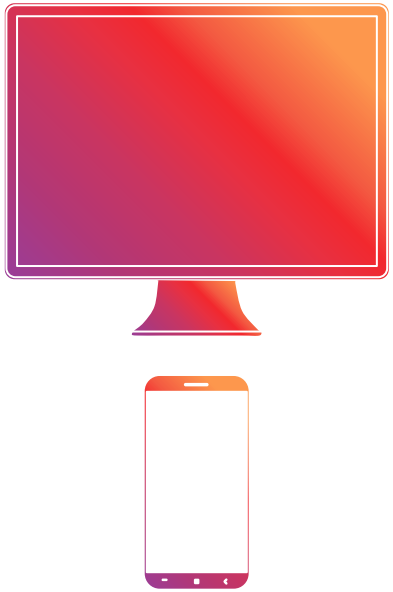
GraphQL actions

Queries
=> fetch data

Mutations
=> create / update / delete data

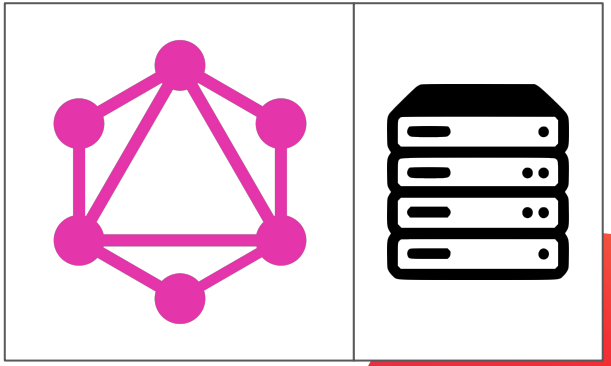
Subscriptions
=> similaire aux queries et mutation ; la souscription permet de créer un tunnel entre le client et le serveur. Le serveur pourra push des informations au client

Ge ry
resources



```
{
  allPersons {
    name
    age
  }
}
```

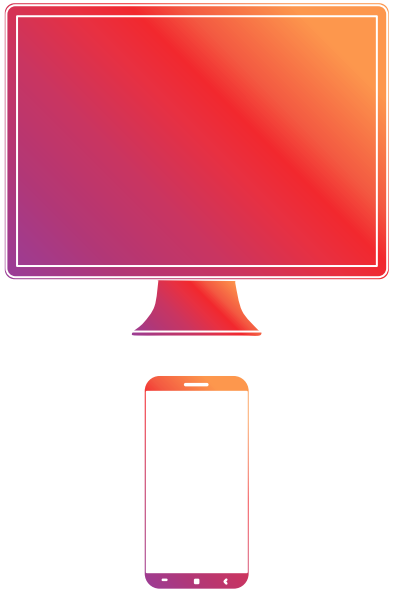
HTTP POST



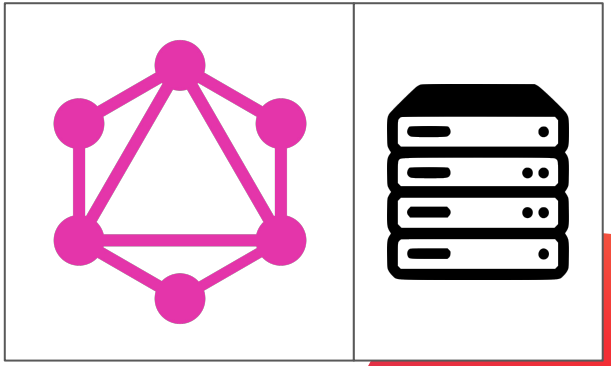
```
{
  "allPersons": [
    { "name": "Albert", age: 49 },
    { "name": "Alphonse", age: 52 },
    { "name": "Alice", age: 29 }
  ]
}
```

G ources

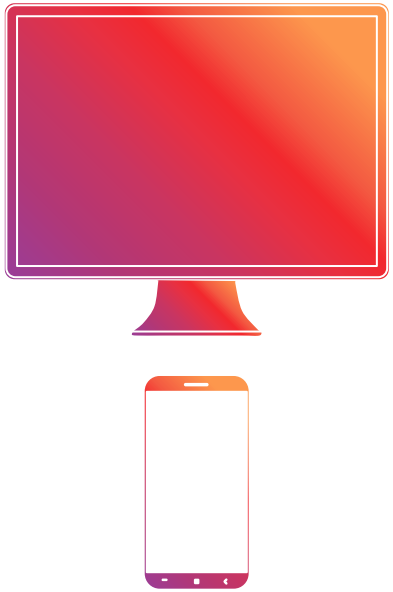
```
{
  allPersons(last: 2) {
    name
    age
  }
}
```



HTTP POST

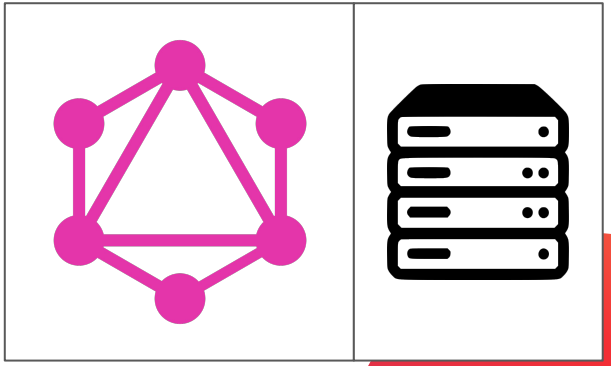


```
{
  "allPersons": [
    { "name": "Alphonse", age: 52 },
    { "name": "Alice", age: 29 }
  ]
}
```



```
mutation {
  createPerson(name: "Robin", age: 32) {
    name
    age
  }
}
```

HTTP POST



```
{
  "createPerson": {
    "name": "Robin",
    "age": 32
  }
}
```

Bénéfices des Schemas & Types

- GraphQL utilise un système de typage fort pour définir les capacités de l'API
- Le schéma sert de contrat entre le back-end et le front-end
- Les équipes back et front peuvent travailler en total indépendances les uns des autres

Query Type

```
{
  allPersons {
    name
    age
  }
}
```

```
type Query {
  allPersons(last: Int): [Person!]!
}
```

Mutation Type

```
mutation {
  createPerson(name: "Robin", age: 32) {
    name
    age
  }
}
```

```
type Mutation {
  createPerson(name: String!, age: Int!): Person!
}
```

Schema complet

```
type Person {
  id: ID!
  name: String!
  age: Int!
  posts: [Posts!]!
}

type Post {
  id: ID!
  title: String!
  author: Person!
}
```

```
type Query {
  allPersons(last: Int): [Person!]!
  allPosts(last: Int): [Post!]!
}
```

```
type Mutation {
  createPerson(name: String!, age: Int!): Person!
  updatePerson(id: ID!, name: String!, age: Int!): Person!
  deletePerson(id: ID!): Person!
  createPost(title: String!): Post!
  updatePost(id: ID!, title: String!): Post!
  deletePost(id: ID!): Post!
}
```

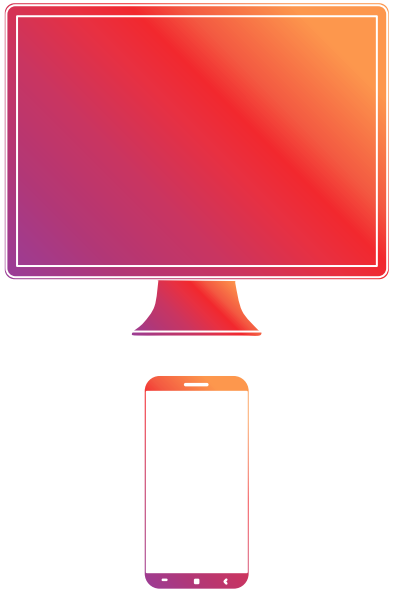
GraphQL resolvers

GraphQL queries/mutation consiste à setter les fields

GraphQL server à une fonction resolver par field

L’objectif de chacun des resolvers est de retrouver la data correspondante au field

Resolvers functions



HTTP POST

```
query {
  User(id: "euxn4kan56l") {
    name
    friends(first: 5) {
      name
      age
    }
  }
}
```

Resolvers

User(id: ID!): User

name(user: User): String!

age(user: User): Int!

friends(first: Int, user: User!):[User!]!

Manipulation GraphQL

https://fakeql.com/graphql/8774836b908ebc4ca7c7ac3840f39199

04

Prisma

And the magic begin

Setup

```
$ ~ mkdir ~/prisma_course
$ ~ cd ~/prisma_course/
$ ~ npm init
```

```
$ ~ npm i graphql-yoga nodemon
```

```
# package.json

{
 ...
 "scripts": {
    "start": "nodemon -e js,graphql src/index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  }
 ...
}
```

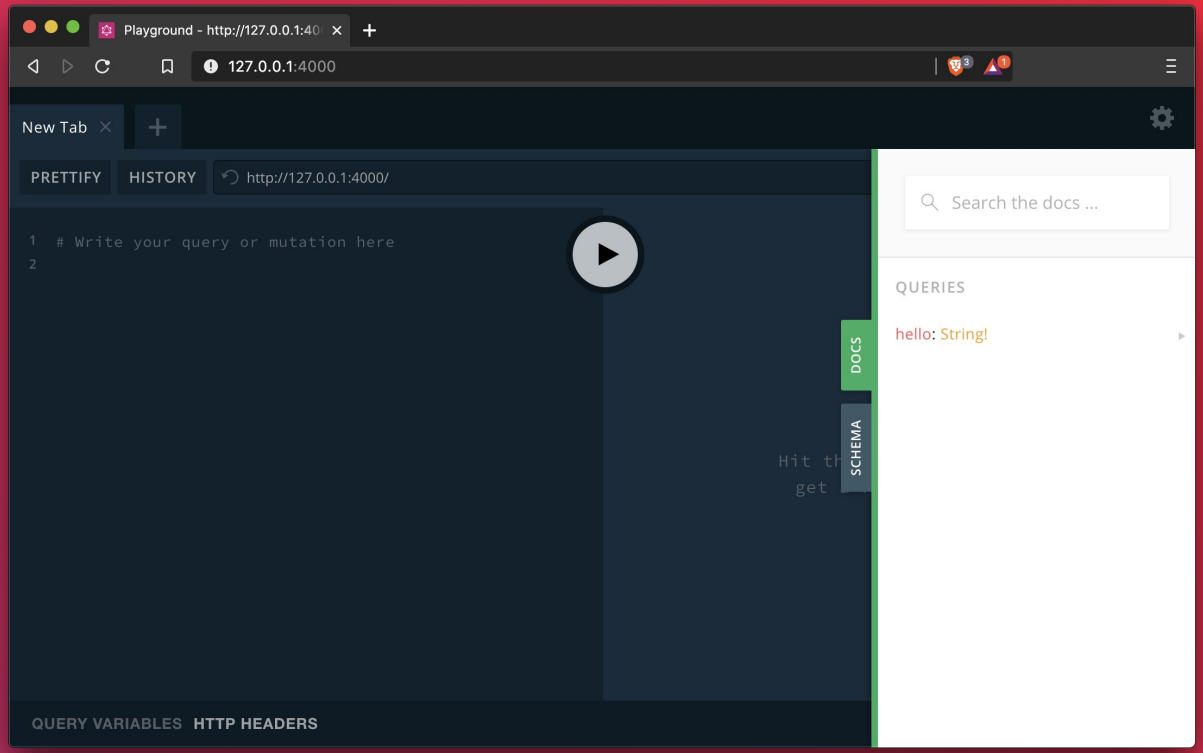
Setup

```
// /src/index.js

const { GraphQLServer } = require('graphql-yoga')

const typeDefs = `
  type Query {
    hello: String!
  }
`

const resolvers = {
    Query: {
        hello: (_, args, context, info) => {
            return "Hello World";
    },
  }
}

const server = new GraphQLServer({ typeDefs, resolvers })
server.start(() => console.log('Server is running on localhost:4000'))
```

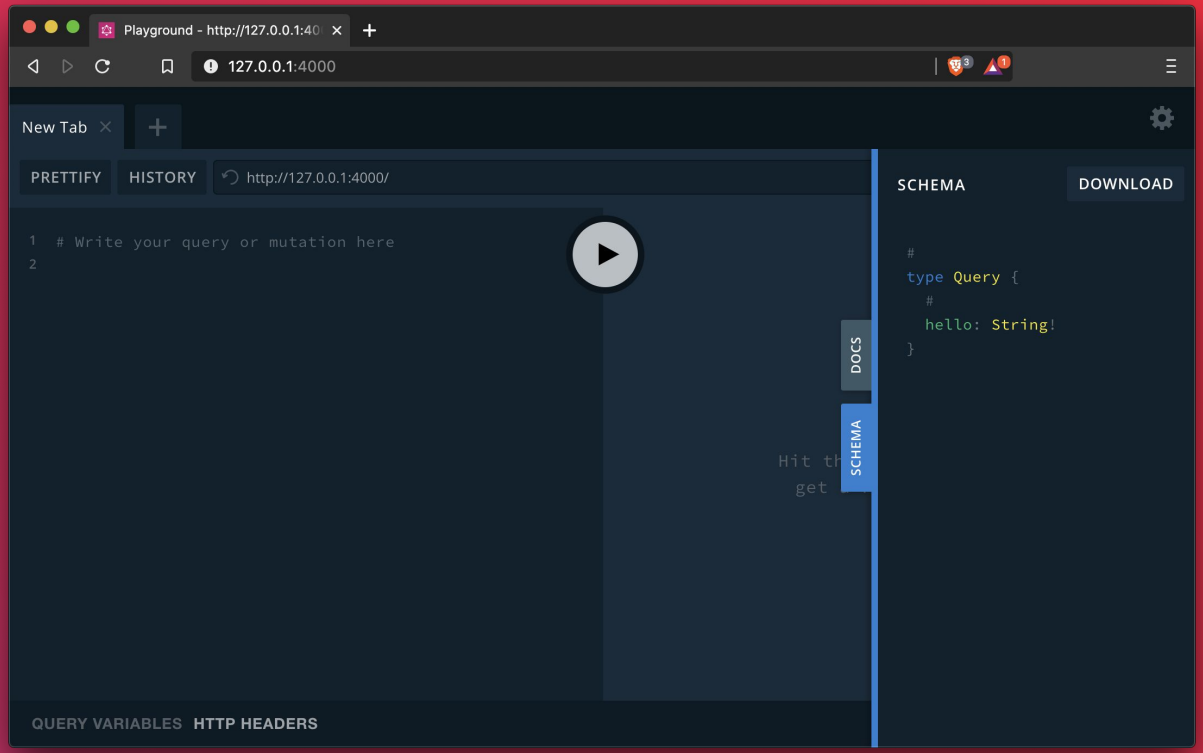
Test

```
$ ~ npm start

> ref@1.0.0 start ~/prisma_course
> nodemon -e js,graphql src/index.js

[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,graphql
[nodemon] starting `node src/index.js`
Server is running on localhost:4000
```

Playground


Playground - http://127.0.0.1:40
127.0.0.1:4000
New Tab
PRETTIFY
HISTORY
http://127.0.0.1:4000/
Write your query or mutation here
Search the docs ...
QUERIES
hello: String!
DOCS
SCHEMA
QUERY VARIABLES
HTTP HEADERS

Playground


Playground - http://127.0.0.1:40
127.0.0.1:4000
New Tab
PRETTIFY
HISTORY
http://127.0.0.1:4000/
Write your query or mutation here
DOCS
SCHEMA
Hit th
get
QUERY VARIABLES
HTTP HEADERS
SCHEMA
DOWNLOAD
#
type Query {
#
hello: String!
}

Query

Playground - http://127.0.0.1:40

127.0.0.1:4000

hello

PRETTIFY HISTORY http://127.0.0.1:4000/ COPY CURL

```
# Write your query or mutation here
query {
  hello
}
```

```
{
  "data": {
    "hello": "Hello World"
  }
}
```

DOCS

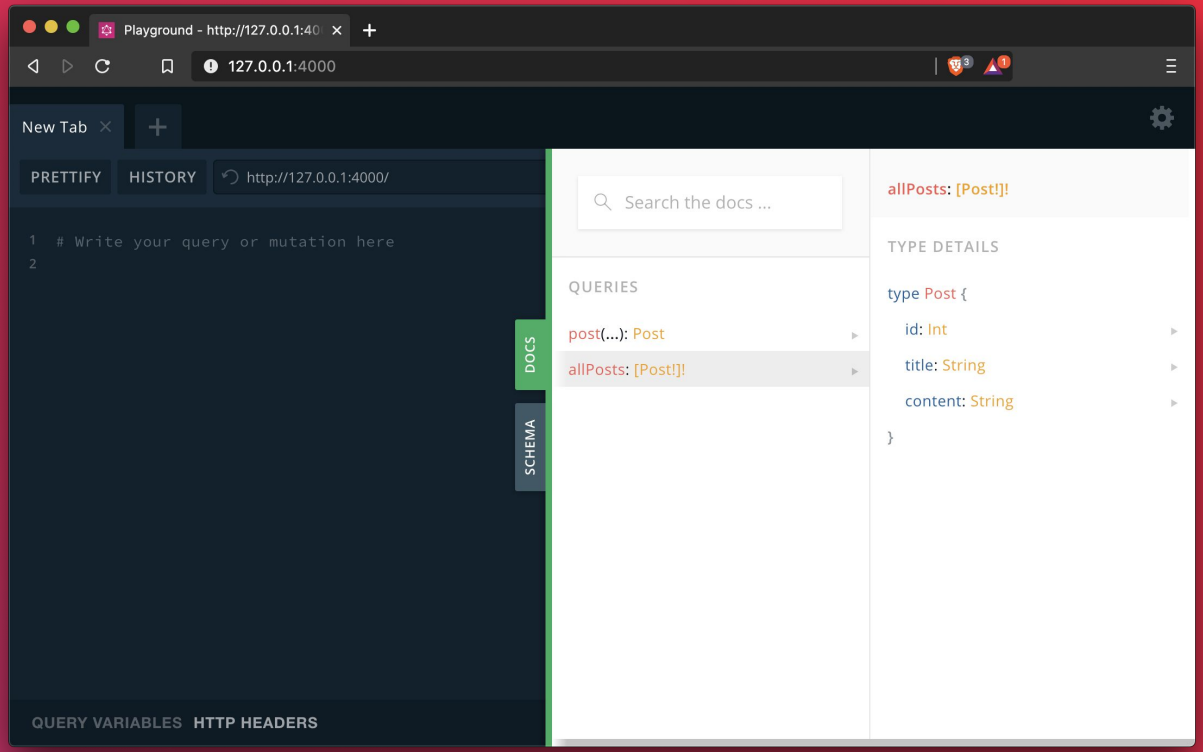
SCHEMA

QUERY VARIABLES HTTP HEADERS TRACING

TypeDef & Resolver

```
// index.js
...

const posts = [
  {
    id: 1,
    title: 'Article one',
    content: 'This is Article One'
  },
  ...
];

const typeDefs = `
 type Query {
  post(postId: Int!): Post
  allPosts: [Post!]!
}

type Post {
   id: Int
   title: String
   content: String
 }
`
const resolvers = {
 Query: {
    post: (_, { postId }, context, info) => {
     return posts.filter(post => post.id === postId)[0];
    },
    allPosts: (_, args, context, info) => {
     return posts;
    }
  },
}

...
```

GraphQL documentation



GraphQL documentation

Playground - http://127.0.0.1:40

127.0.0.1:4000

New Tab

PRETTIFY HISTORY http://127.0.0.1:4000/

```
# Write your query or mutation here
```

QUERY VARIABLES HTTP HEADERS

DOCS

SCHEMA

SCHEMA DOWNLOAD

```
#
type Query {
  #
  post(postId: Int!): Post

  #
  allPosts: [Post!]!
}

#
type Post {
  #
  id: Int

  #
  title: String

  #
  content: String
}
```

Setting up Prisma

```
$ ~ npm i -g prisma1
$ ~ npm i dotenv
$ ~ prisma1 init
$ ~ docker-compose up -d
```

```yaml
# prisma.yml

endpoint: ${env:URL_DB_PRISMA}
datamodel: datamodel.prisma

generate:
  - generator: javascript-client
    output: ./generated/prisma-client/

hooks:
  post-deploy:
    - echo "Deployment finished"
    - graphql get-schema --project prisma
    - graphql codegen
```