

GraphQL 粗览

芋头 @ 预策科技



什么是 GraphQL?

GraphQL 是一种针对 Graph (图状数据) 进行查询特别有优势的 Query Language (查询语言)

它跟 SQL 的关系是共用 QL 后缀，就好像「汉语」和「英语」共用后缀一样，但他们本质上是不同的语言。GraphQL 跟用作存储的 NoSQL 没有必然联系，虽然 GraphQL 背后的实际存储可以选择 NoSQL 类型的数据库，但也可以用 SQL 类型的数据库，或者任意其它存储方式 (例如文本文件、存内存里等等)

GraphQL 最大的优势是查询图状数据

图状数据查询

假设每个 API 负责请求一种类型的对象，例如用户是一个类型，帖子是另一个类型，那就需要非常多个请求才能把这个页面所需的所有数据拿回来

而且这些请求直接还存在依赖关系，不能平行地发多个请求，例如说在获得帖子数据之前，无法请求评论数据；在获得评论数据之后，才能开始请求评论作者数据

GraphQL 能够很好地解决这个问题，但前提是数据已经以图的数据结构进行保存。例如上面说到的用户、帖子、评论是顶点，而用户跟用户发过的帖子存在边的关系，帖子跟帖子评论存在一对多的边，评论跟评论作者存在一对一的边。这时候如果新产品引入了新的对象类型（也就是顶点类型）和新的边类型，那没有关系。在查询数据时用 GraphQL 描述一下要查询的这些边和顶点就行，不需要去改 API 实现

```
1 我的名字
2 我的头像
3 我的好友（按他们跟你的亲疏程度排序取前 6）：
4 * 好友 1 的名字、头像及链接
5 * 好友 2 的名字、头像及链接
6 * .....
7 我的照片（按时间倒序排序取前 6）：
8 * 照片 1 及其链接
9 * 照片 2 及其链接
10 * .....
11 我的帖子（按时间倒序排序）：
12 * 帖子 1：
13     * 帖子 1 内容
14     * 帖子 1 评论：
15         * 帖子 1 评论 1：
16             * 帖子 1 评论 1 内容
17             * 帖子 1 评论 1 作者名字
18             * 帖子 1 评论 1 作者头像
19     * 帖子 1 评论 2：
20         * .....
21     * .....
22 * 帖子 2：
23     * 帖子 2 内容
24     * 帖子 2 评论：
25         * .....
26 * .....
27
```



描述你的数据

```
type Project {  
  name: String!  
  tagline: String!  
  contributors: [User]  
}
```

请求你所要的数据

```
{  
  project(name: "GraphQL") {  
    tagline  
  }  
}
```

得到可预测的结果

```
{  
  "project": {  
    "tagline": "A query language for APIs"  
  }  
}
```

核心特性

请求你所要的数据

向你的 API 发出一个 GraphQL 请求就能准确获得你想要的数据，不多不少。GraphQL 查询总是返回可预测的结果。使用 GraphQL 的应用可以工作得又快又稳，因为控制数据的是应用，而不是服务器。

只用一个请求，获取多个资源

GraphQL 查询不仅能够获得资源的属性，还能沿着资源间引用进一步查询。典型的 REST API 请求多个资源时得载入多个 URL，而 GraphQL 可以通过一次请求就获取你应用所需的所有数据。这样一来，即使是比较慢的移动网络连接下，使用 GraphQL 的应用也能表现得足够迅速。

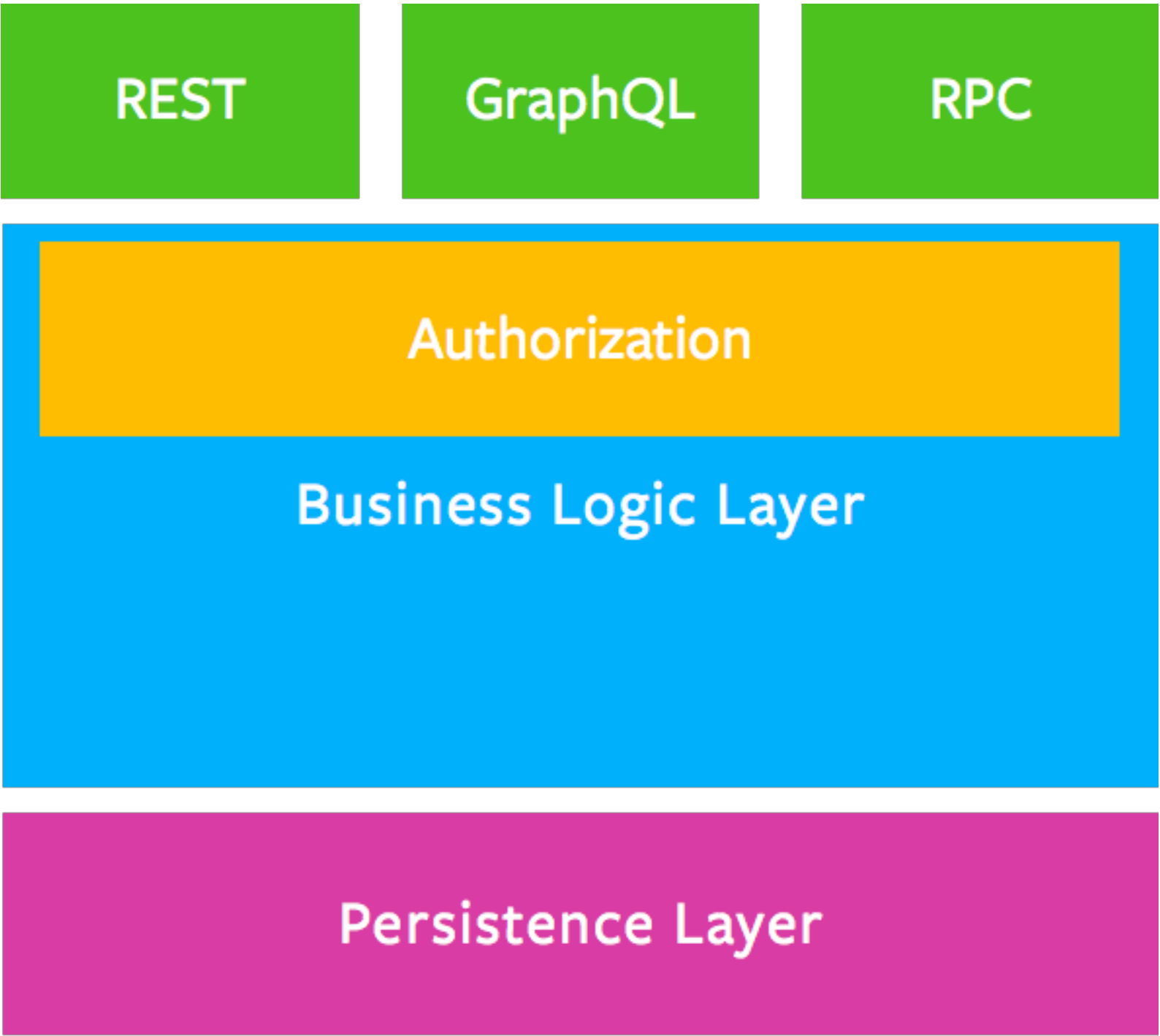
描述所有的可能类型系统

GraphQL 查询不仅能够获得资源的属性，还能沿着资源间引用进一步查询。典型的 REST API 请求多个资源时得载入多个 URL，而 GraphQL 可以通过一次请求就获取你应用所需的所有数据。这样一来，即使是比较慢的移动网络连接下，使用 GraphQL 的应用也能表现得足够迅速。

*Nam liber tempor cum soluta nobis eleifend option congue nihil
imperdiet doming id quod mazim*

典型架构

系统中的所有入口点（REST、GraphQL和RPC）都将使用相同的验证、授权和错误处理规则进行处理。



查询示例

获取我所有帐户的收件箱里未读邮件的数量

```
1 {  
2   accounts {  
3     inbox {  
4       unreadEmailCount  
5     }  
6   }  
7 }
```

carbon
carbon.now.sh

获取主账户的前二十封草稿邮件的“预览信息”


```
1 {  
2   mainAccount {  
3     drafts(first: 20) {  
4       ...previewInfo  
5     }  
6   }  
7 }  
8  
9 fragment previewInfo on Email {  
10  subject  
11  bodyPreviewSentence  
12 }
```

carbon
carbon.now.sh

操作


QUERY 查询

```
1 query MyQuery {  
2   queryFeature(arg1: {unitTypeList: 1})  
3 {   name  
4     uid  
5 }  
6 }
```



MUTATION (突变)

```
1 mutation MyMutation2 {  
2   updateProject(input: {projectId: ""}) {  
3     clientMutationId  
4   }  
5 }
```

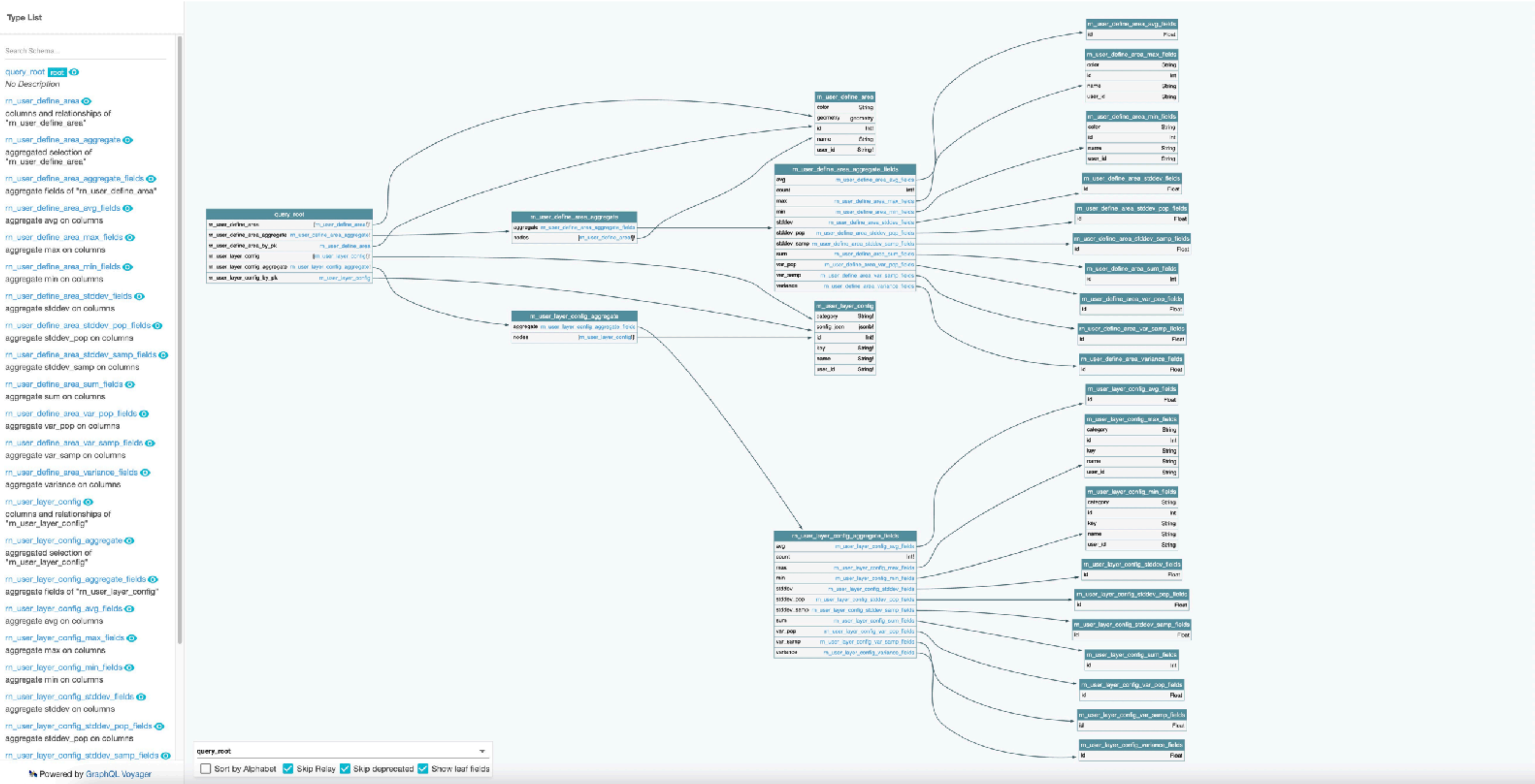


SUBSCRIPTION (订阅)

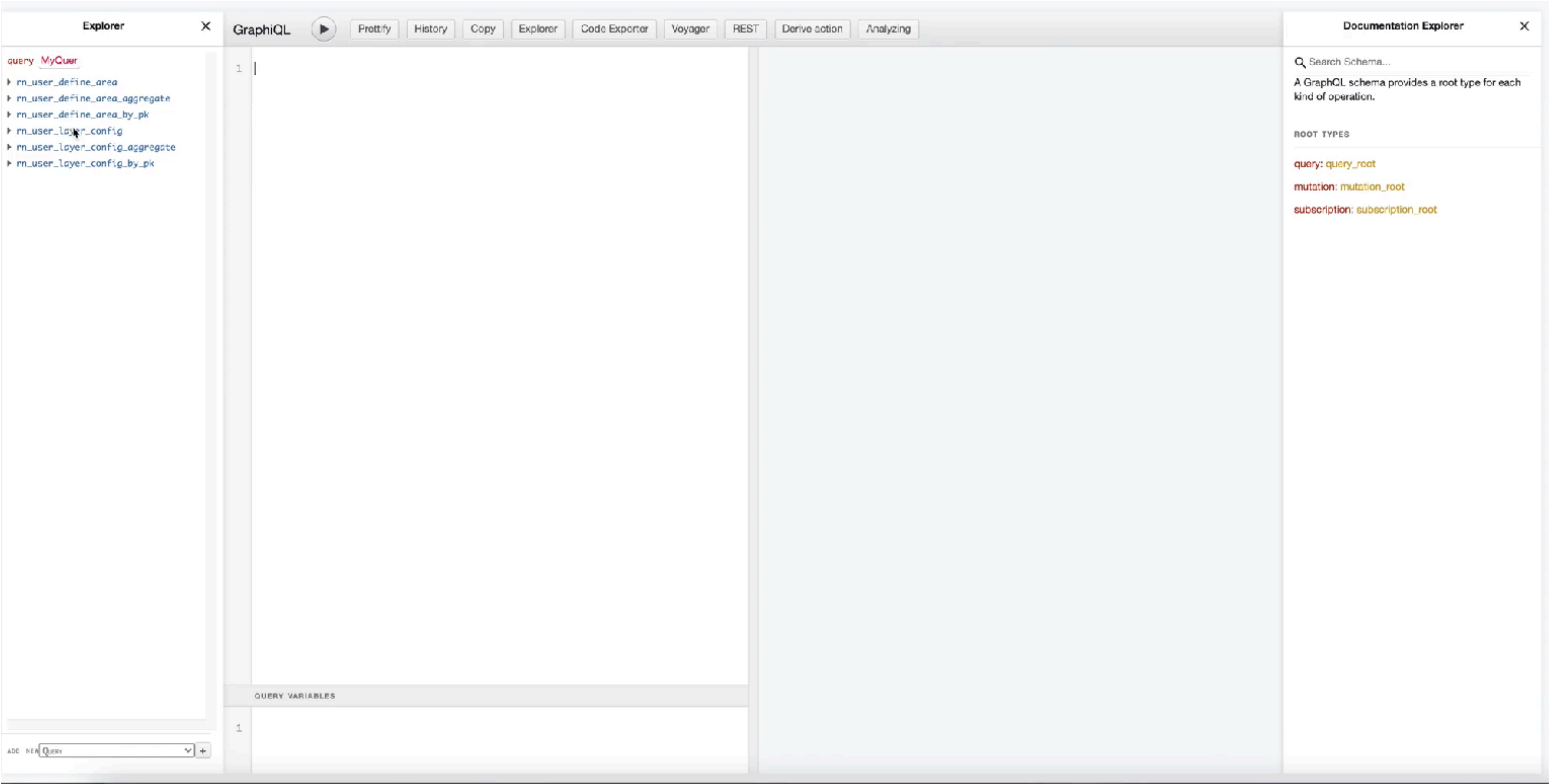
```
1 subscription MySubscription {  
2   rn_user_layer_config(limit: 10, where: {})  
3 {   name  
4     user_id  
5 }  
6 }  
7
```



工具

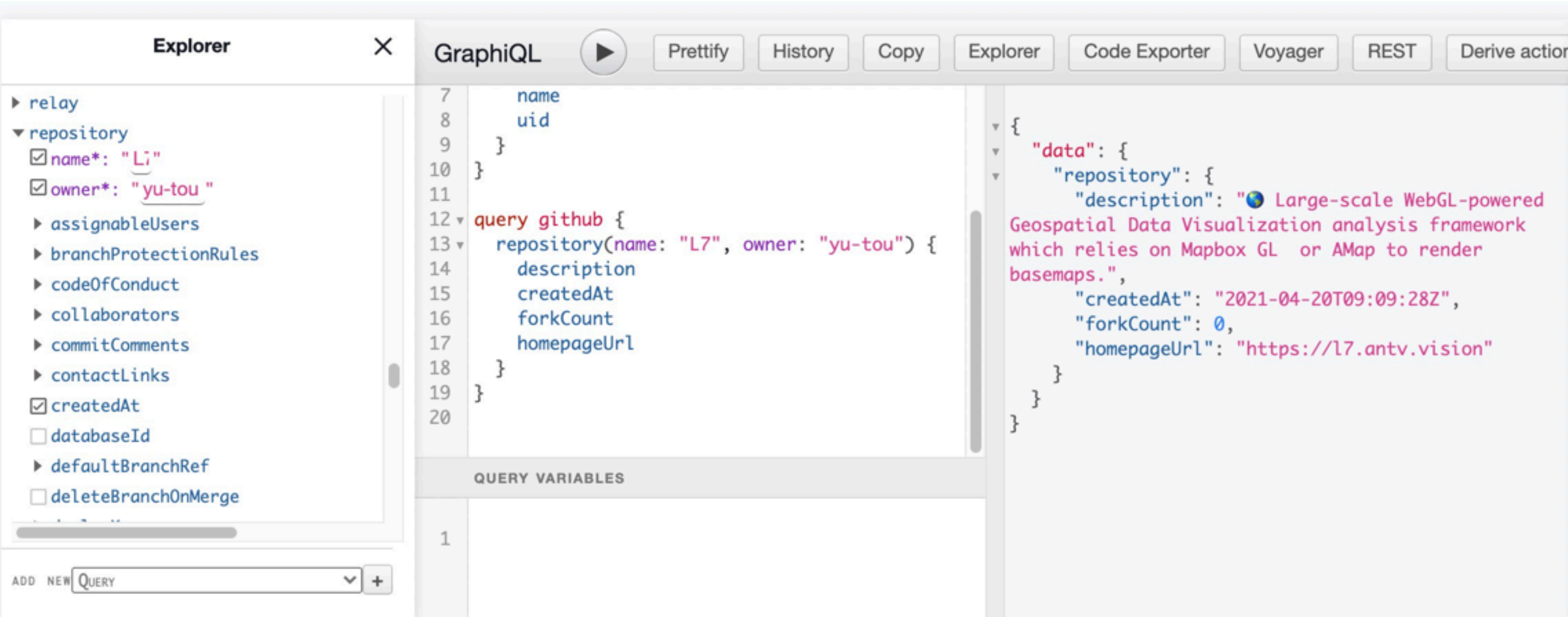


<https://github.com/graphql/graphiql>



开放

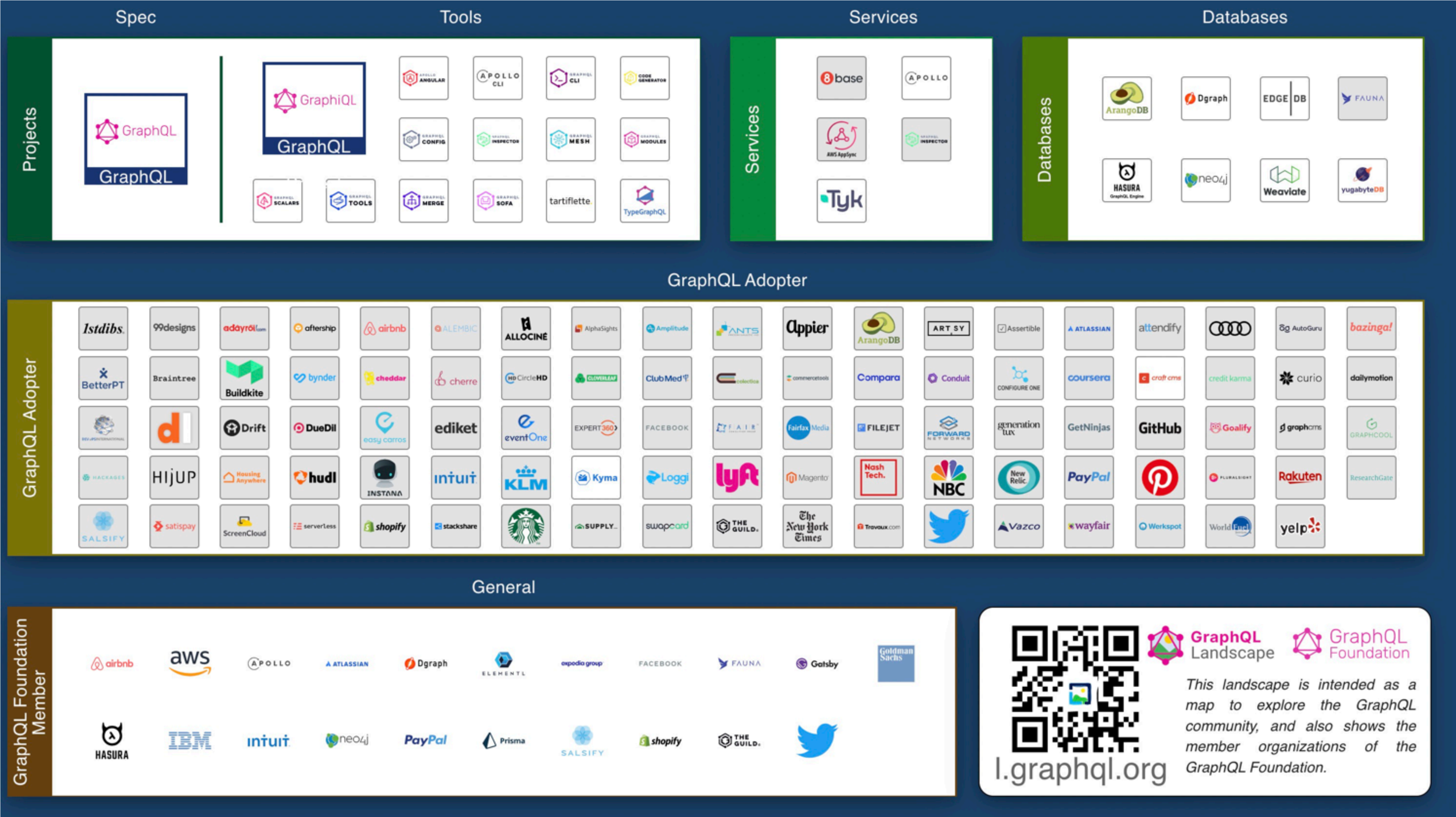
<https://api.github.com/graphql>



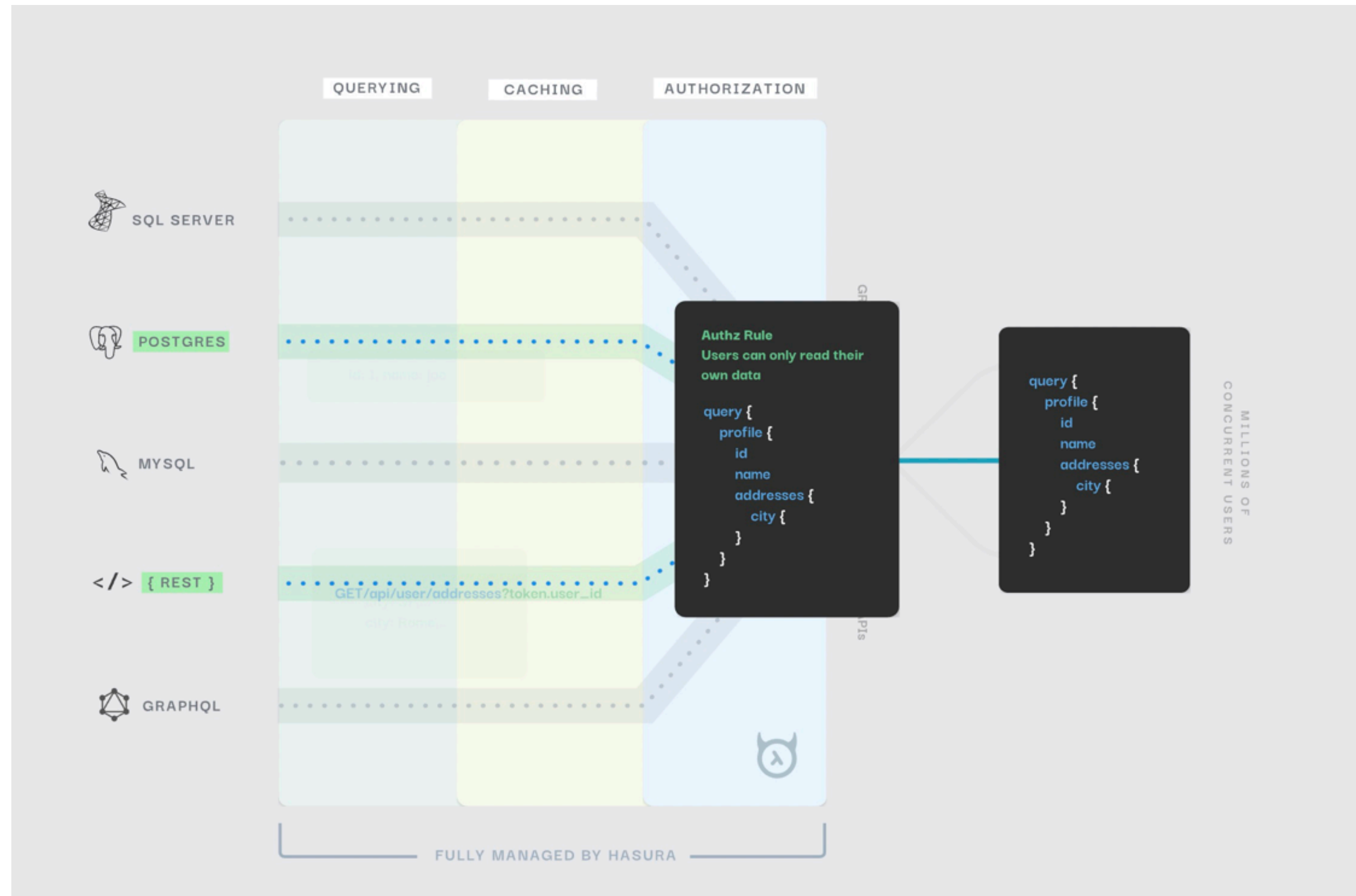
为什么选择 GraphQL 做服务开放

1. 服务隔离，强解耦（服务->声明->用户）
2. 从声明或者代码生成所有用户侧信息（文档、示例、mock）
3. 想要什么就拿什么，减少冗余字段
4. 想要什么就拿什么，多个节点分布一层一层获取，而不是固定的树形结构 api
5. 一次请求，所有数据

生态



今日主角 Hasura



核心能力 Scheme



Scheme to GraphQL

丰富的
Query 参数
生成

自动生成
增删改查的操作

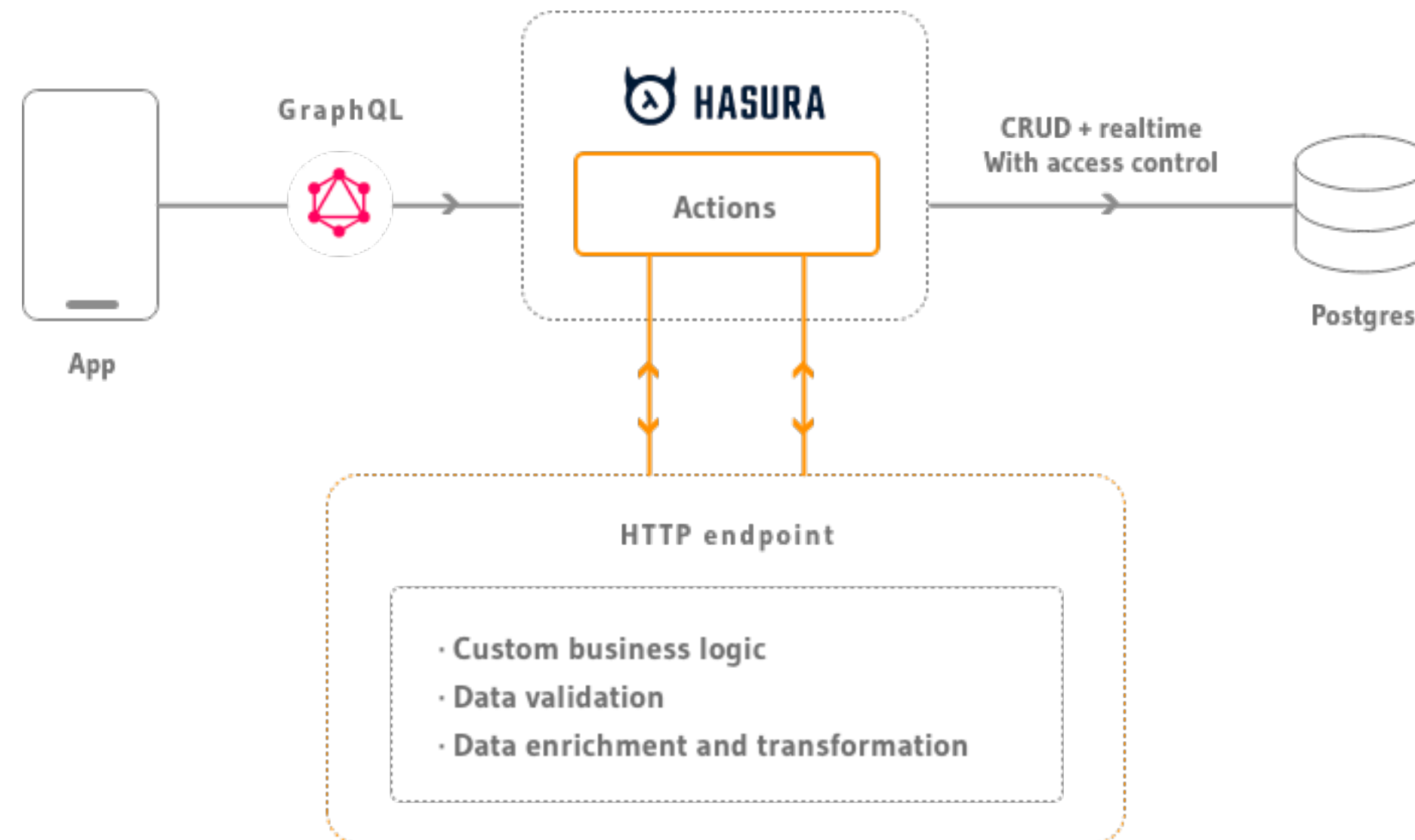
```
resource
  rn_user_define_area
    distinct_on:
    limit: $ 10
    offset: $ 10
    order_by:
      color:
      geometry:
      id: asc
      name:
      user_id:
    where:
      _and:
      _not:
      _or:
      color:
      geometry:
      id:

4
5 query MyQuery {
6   queryFeature(arg1: {unitTypeList: 1}) {
7     name
8     uid
9   }
10 }
11
12 query github {
13   rn_user_define_area(limit: 10, offset: 10, order_by: {id: asc}, where: {id: {_eq: 10}}) {
14     geometry
15     id
16     name
17   }
18 }
19
```

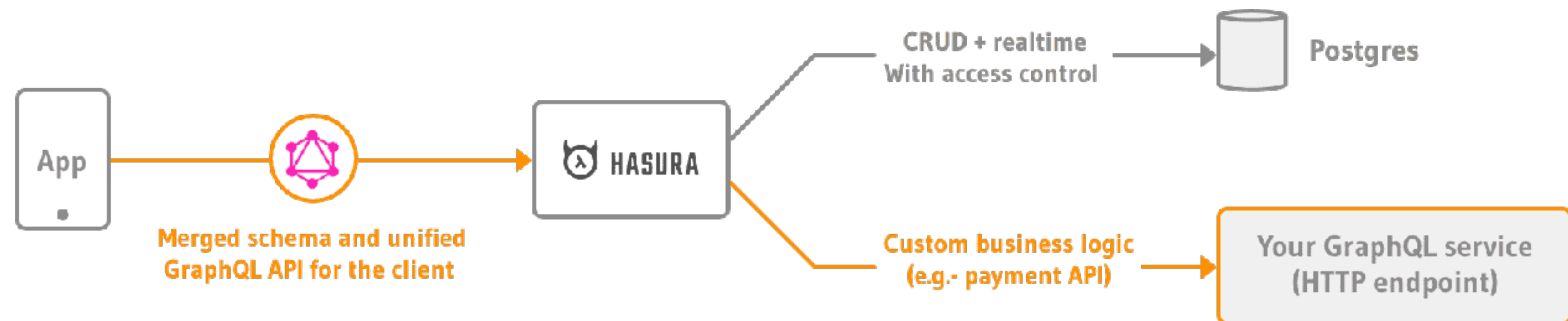
- deleteTeamDiscussion
- deleteTeamDiscussionComment
- deleteVerifiableDomain
- delete_rn_user_define_area
- delete_rn_user_define_area_by_pk
- delete_rn_user_layer_config
- delete_rn_user_layer_config_by_pk
- disablePullRequestAutoMerge
- dismissPullRequestReview
- enablePullRequestAutoMerge
- followUser
- insert_rn_user_define_area
- insert_rn_user_define_area_one
- insert_rn_user_layer_config

Rest & GraphQL to GraphQL

Restful
To
GraphQL



GraphQL
To
GraphQL



Database Events Trigger

Create a new event trigger

Trigger Name ⓘ

Database ⓘ

Schema/Table ⓘ

Trigger Operations ⓘ
☒ Insert ☒ Update ☐ Delete ☐ Via console ⓘ [\(Know more\)](#)

Webhook URL ⓘ

URL ▼

Note: Specifying the webhook URL via an environmental variable is recommended if you have different URLs for multiple environments.



THANK YOU

芋头 @ 预策科技

sunxinyu@yucekj.com