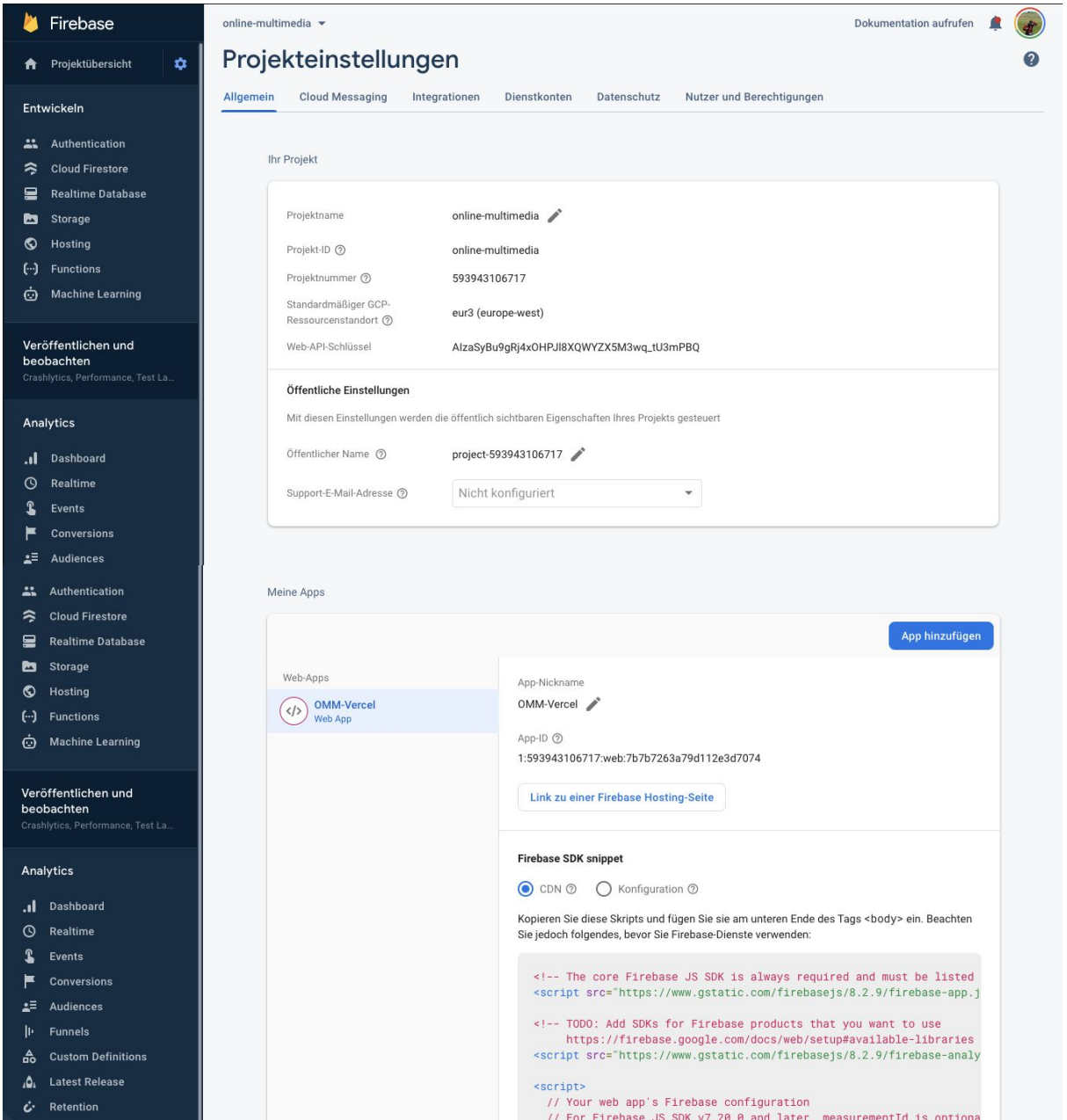
Firebase

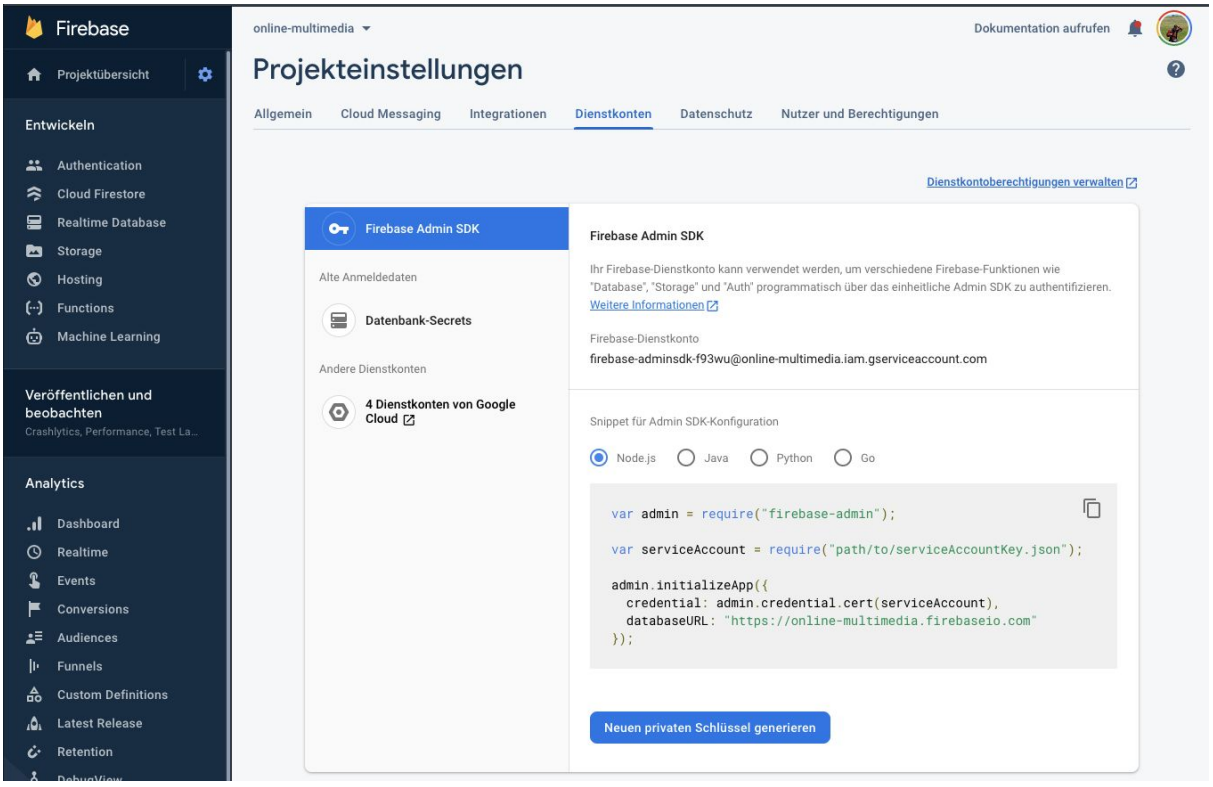
As seen in the next screenshot we use `europe-west3` as a data/project-location, which corresponds to Frankfurt, Germany.
In order to connect firebase to our app you have to get the configuration & admin configuration data and write in into the corresponding fields in the .env.local file you can get these as follows:

Configuration:

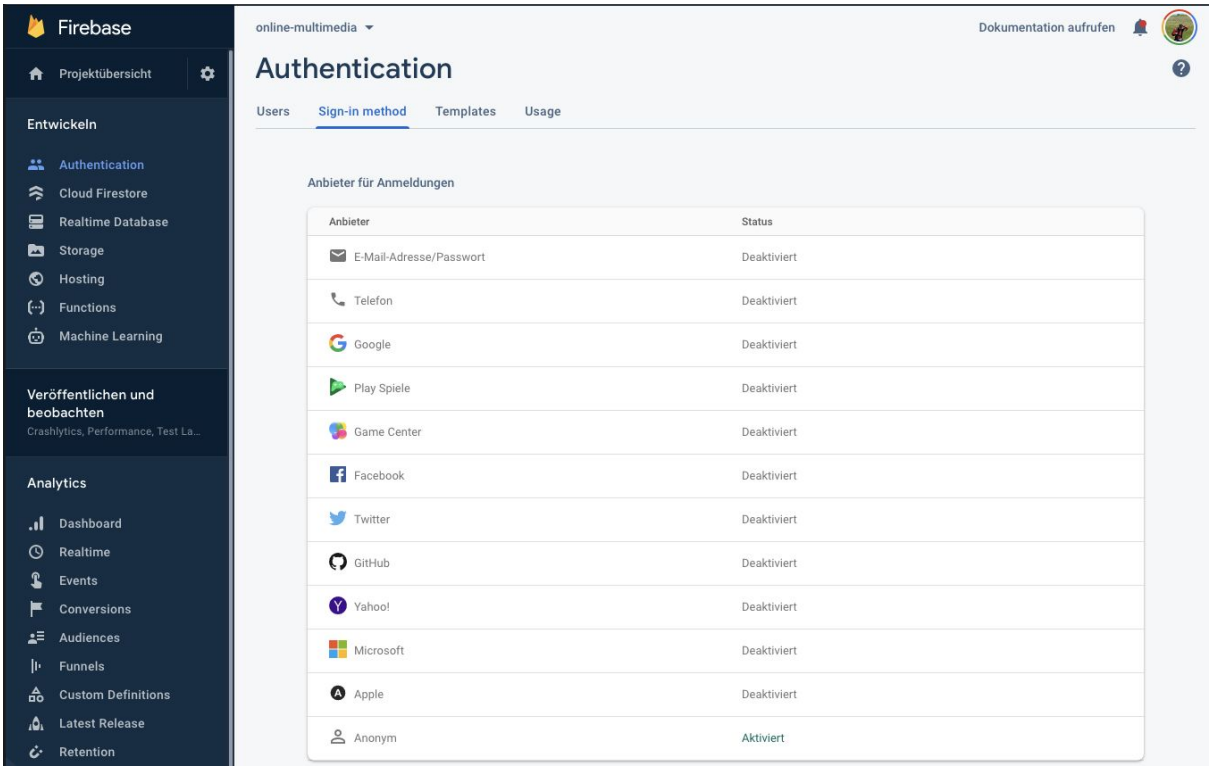


Admin configuration:

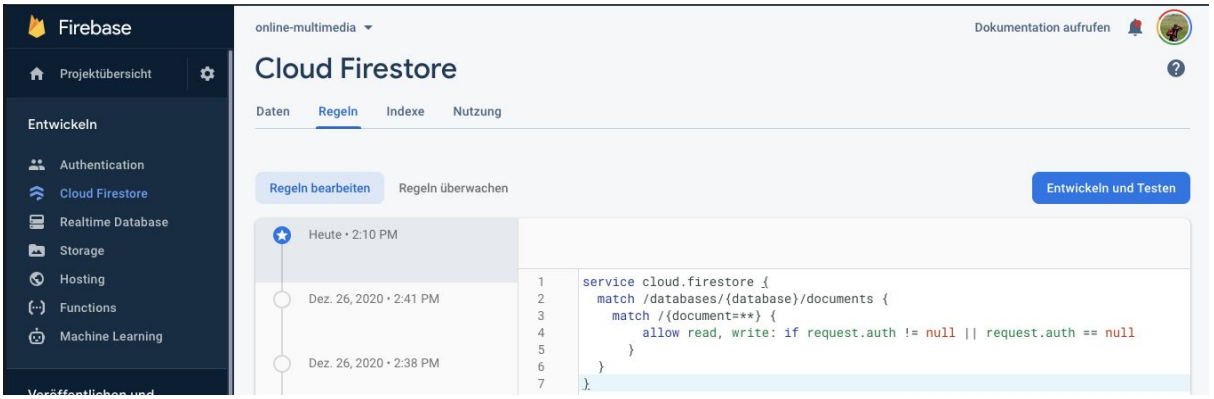
Click “Neuen schlüssel generieren” and get the .env fields from the JSON you downloaded



Authentication Settings (->Activate “anonym”)



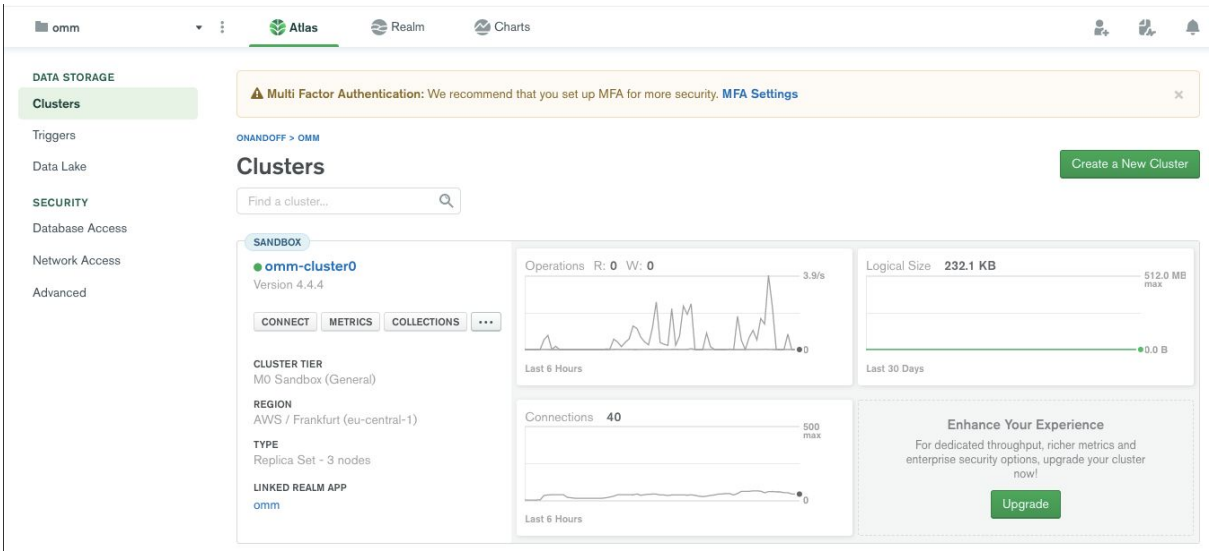
Cloud Firestore Rules



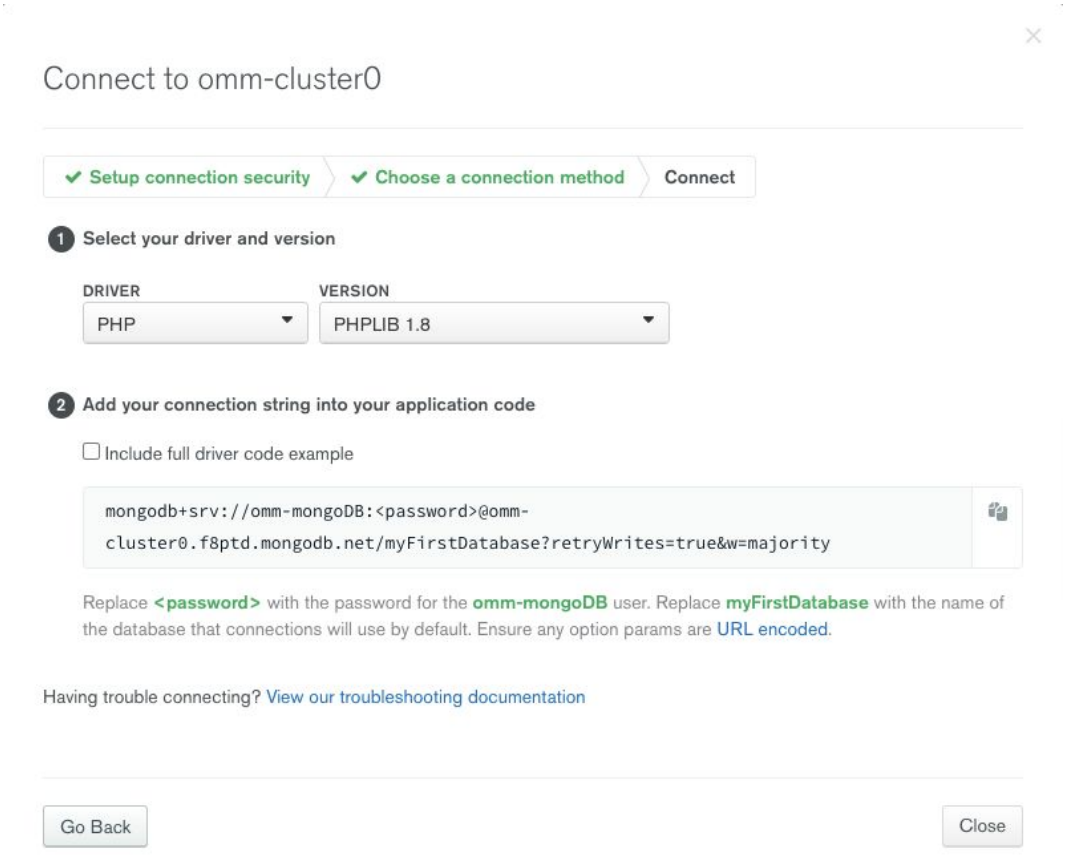
MongoDB Atlas & Realm

As seen in the next screenshot we use AWS / Frankfurt (eu-central-1) as a data/project-location, which corresponds to Frankfurt, Germany.

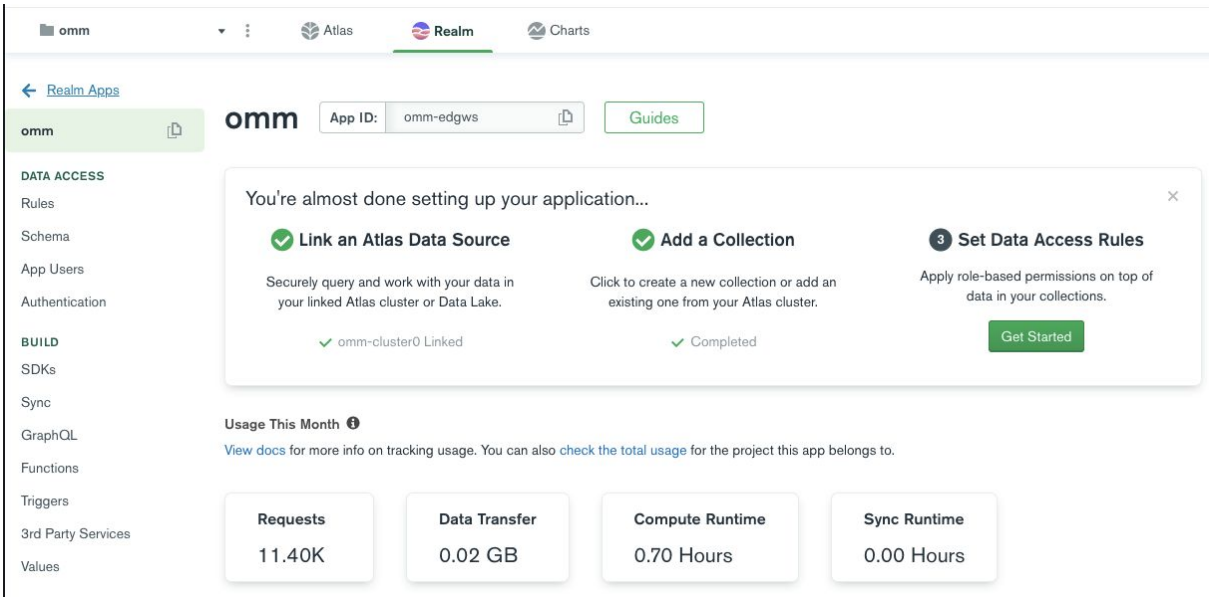
Get the **NEXT_PUBLIC_MONGODB_URI**:



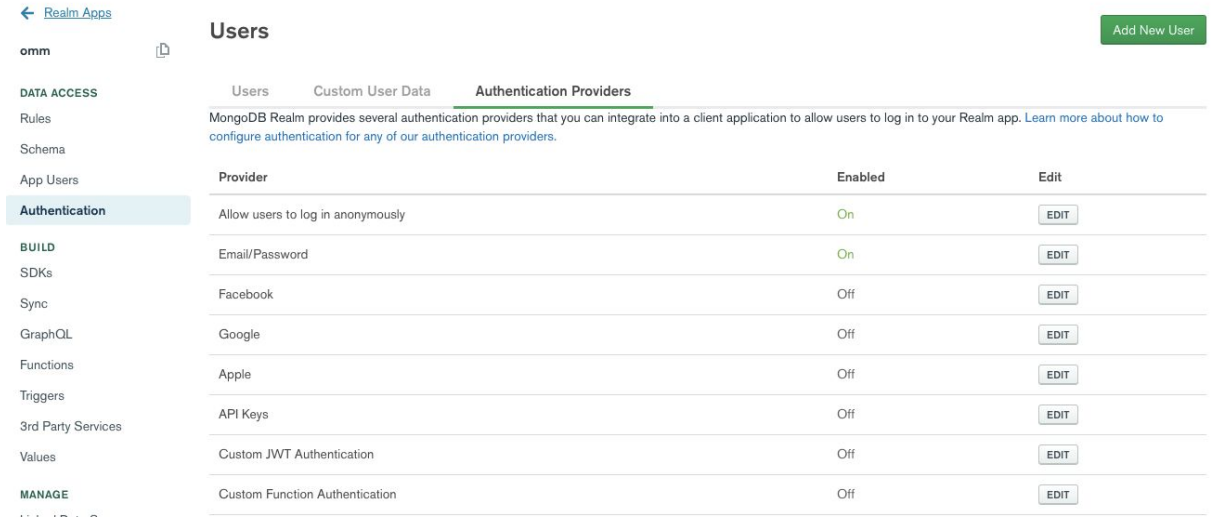
Click on “CONNECT” (remove “myFirstDatabase” from the url & use your user and password):



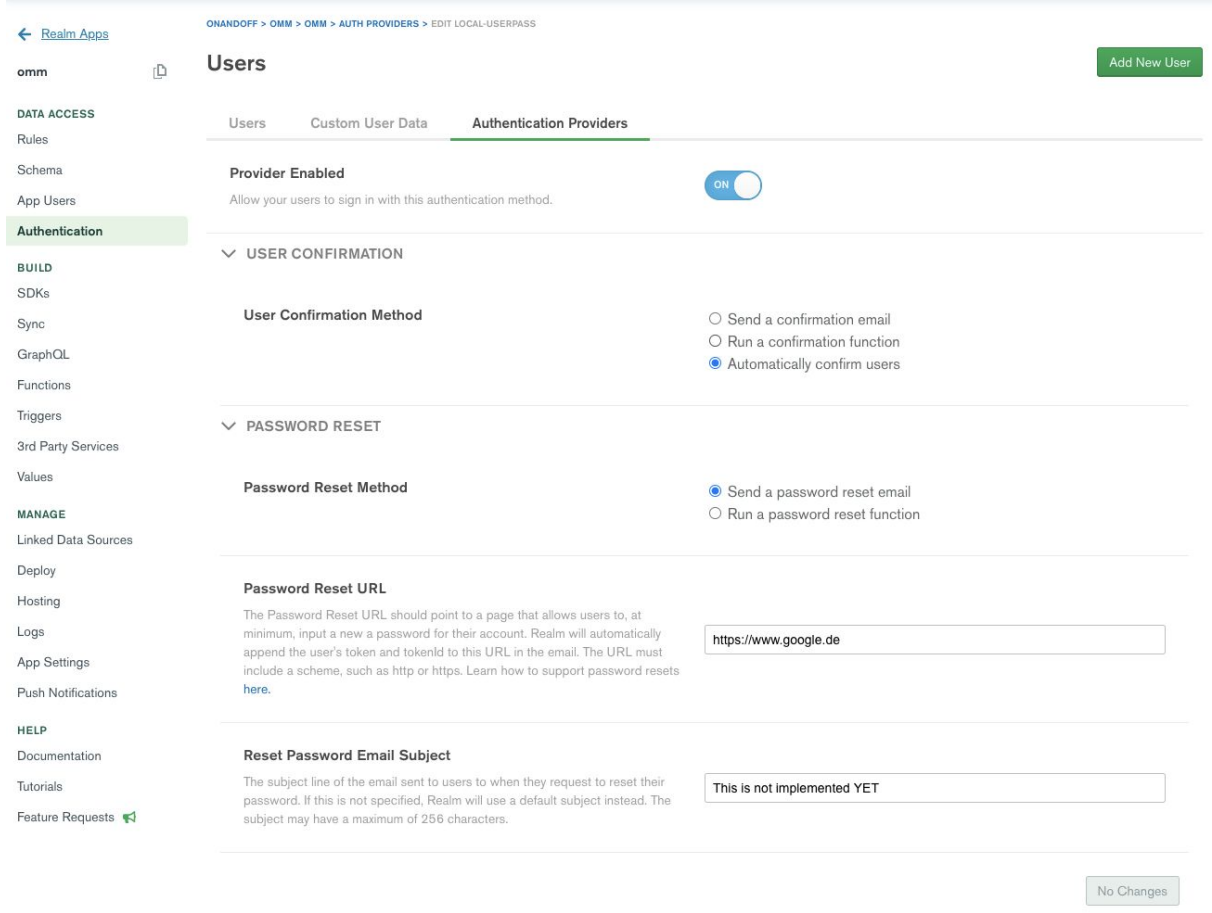
Get **NEXT_PUBLIC_MONGODB_REALM_ID** here (App ID):



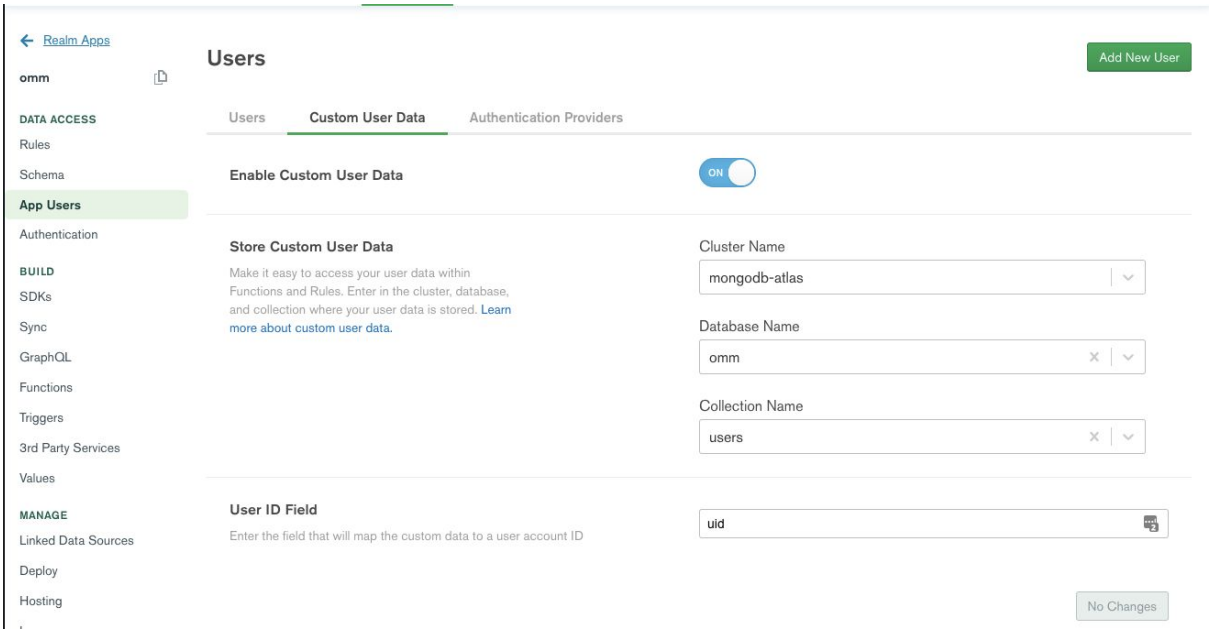
Authentication & Users



Email/Password Settings

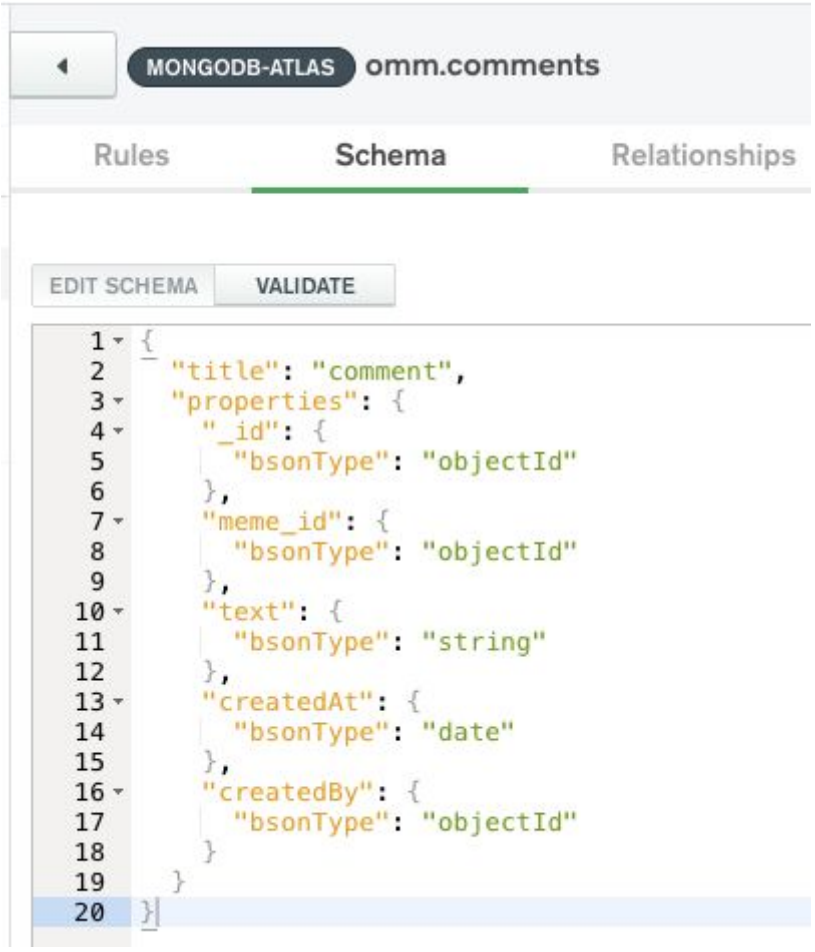


Custom User Data

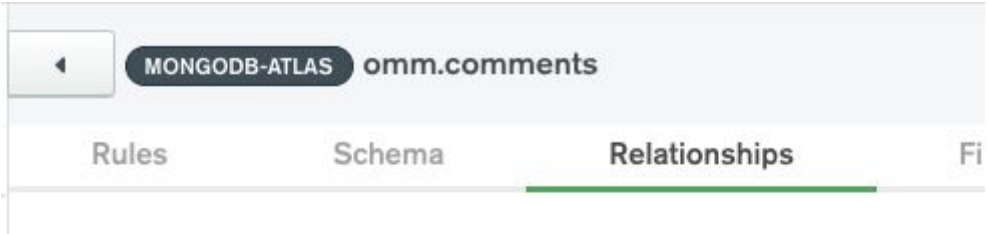


Schemas & Relationships

Comments




```
{
  "title": "comment",
  "properties": {
    "_id": {
      "bsonType": "objectId"
    },
    "meme_id": {
      "bsonType": "objectId"
    },
    "text": {
      "bsonType": "string"
    },
    "createdAt": {
      "bsonType": "date"
    },
    "createdBy": {
      "bsonType": "objectId"
    }
  }
}
```




```
{
  "createdBy": {
    "ref": "#/relationship/mongodb-atlas/omm/users",
    "foreign_key": "_id",
    "is_list": false
  },
  "meme_id": {
    "ref": "#/relationship/mongodb-atlas/omm/memes",
    "foreign_key": "_id",
    "is_list": false
  }
}
```

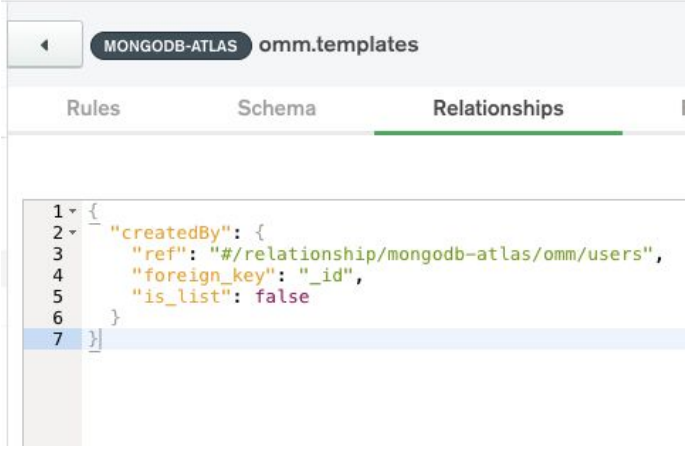
Memes


```
{
  "title": "meme",
  "properties": {
    "_id": {
      "bsonType": "objectId"
    },
    "commentCount": {
      "bsonType": "int"
    },
    "comments": {
      "bsonType": "array",
      "items": {
        "bsonType": "objectId"
      }
    },
    "createdAt": {
      "bsonType": "date"
    },
    "createdBy": {
      "bsonType": "objectId"
    },
    "downVotes": {
      "bsonType": "array",
      "items": {
        "bsonType": "objectId"
      }
    },
    "forkedBy": {
      "bsonType": "array",
      "items": {
        "bsonType": "objectId"
      }
    },
    "forkedFrom": {
      "bsonType": "objectId"
    },
    "isDraft": {
      "bsonType": "bool"
    },
    "json": {
      "bsonType": "string"
    },
    "points": {
      "bsonType": "int"
    },
    "svg": {
      "bsonType": "string"
    },
    "template": {
      "bsonType": "objectId"
    },
    "title": {
      "bsonType": "string"
    },
    "upVotes": {
      "bsonType": "array",
      "items": {
        "bsonType": "objectId"
      }
    },
    "url": {
      "bsonType": "string"
    },
```

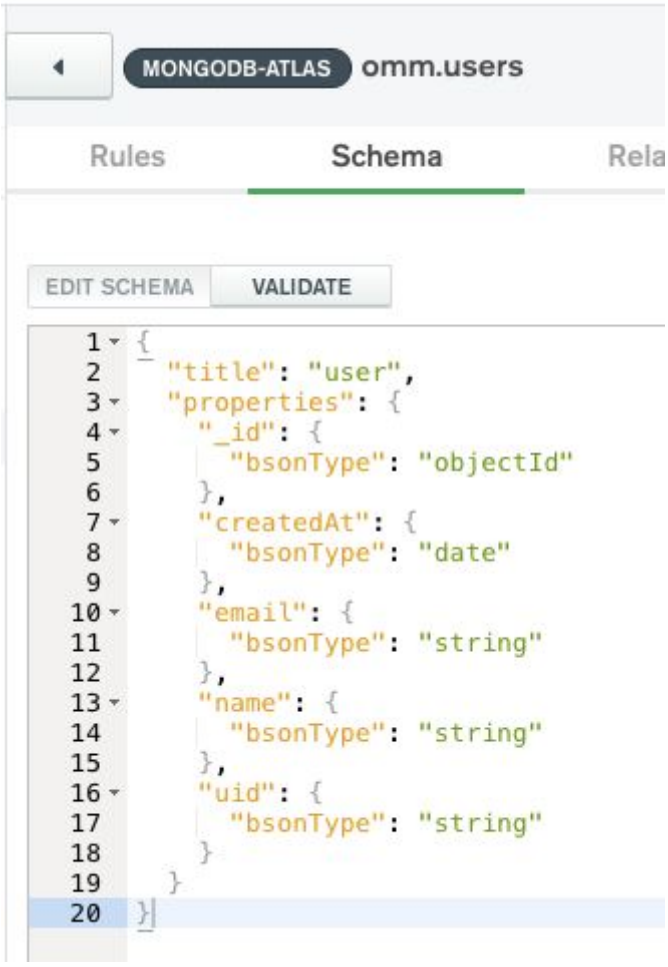

```
{
  "comments": {
    "ref": "#/relationship/mongodb-atlas/omm/comments",
    "foreign_key": "_id",
    "is_list": true
  },
  "createdBy": {
    "ref": "#/relationship/mongodb-atlas/omm/users",
    "foreign_key": "_id",
    "is_list": false
  },
  "downVotes": {
    "ref": "#/relationship/mongodb-atlas/omm/users",
    "foreign_key": "_id",
    "is_list": true
  },
  "forkedBy": {
    "ref": "#/relationship/mongodb-atlas/omm/memes",
    "foreign_key": "_id",
    "is_list": true
  },
  "forkedFrom": {
    "ref": "#/relationship/mongodb-atlas/omm/memes",
    "foreign_key": "_id",
    "is_list": false
  },
  "template": {
    "ref": "#/relationship/mongodb-atlas/omm/templates",
    "foreign_key": "_id",
    "is_list": false
  },
  "upVotes": {
    "ref": "#/relationship/mongodb-atlas/omm/users",
    "foreign_key": "_id",
    "is_list": true
  }
}
```

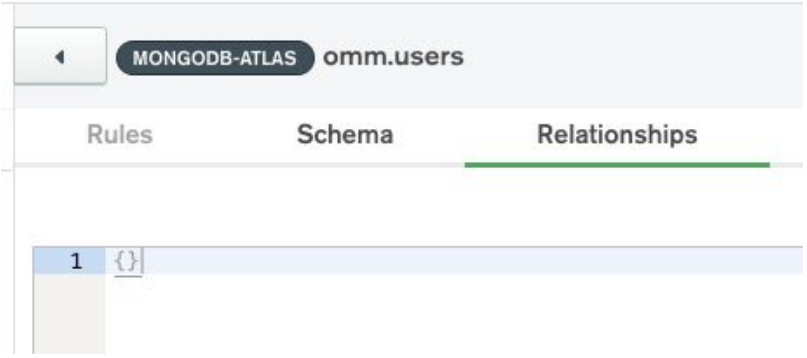
Templates


MONGODB-ATLAS
omm.templates
Rules
Schema
Relationships
EDIT SCHEMA
VALIDATE
{
"title": "template",
"properties": {
"_id": {
"bsonType": "objectId"
},
"createdAt": {
"bsonType": "date"
},
"createdBy": {
"bsonType": "objectId"
},
"height": {
"bsonType": "int"
},
"img": {
"bsonType": "string"
},
"type": {
"bsonType": "string"
},
"mediaType": {
"bsonType": "string"
},
"url": {
"bsonType": "string"
},
"width": {
"bsonType": "int"
},
"duration": {
"bsonType": "number"
},
"name": {
"bsonType": "string"
}
}
}


MONGODB-ATLAS
omm.templates
Rules
Schema
Relationships
{
"createdBy": {
"ref": "#/relationship/mongodb-atlas/omm/users",
"foreign_key": "_id",
"is_list": false
}
}

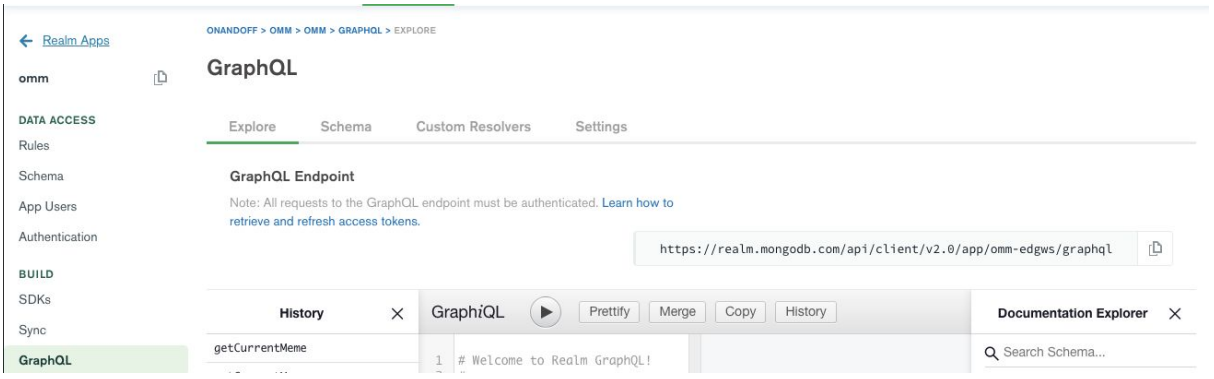
Users


MONGODB-ATLAS
omm.users
Rules
Schema
Rela
EDIT SCHEMA
VALIDATE
{
"title": "user",
"properties": {
"_id": {
"bsonType": "objectId"
},
"createdAt": {
"bsonType": "date"
},
"email": {
"bsonType": "string"
},
"name": {
"bsonType": "string"
},
"uid": {
"bsonType": "string"
}
}
}

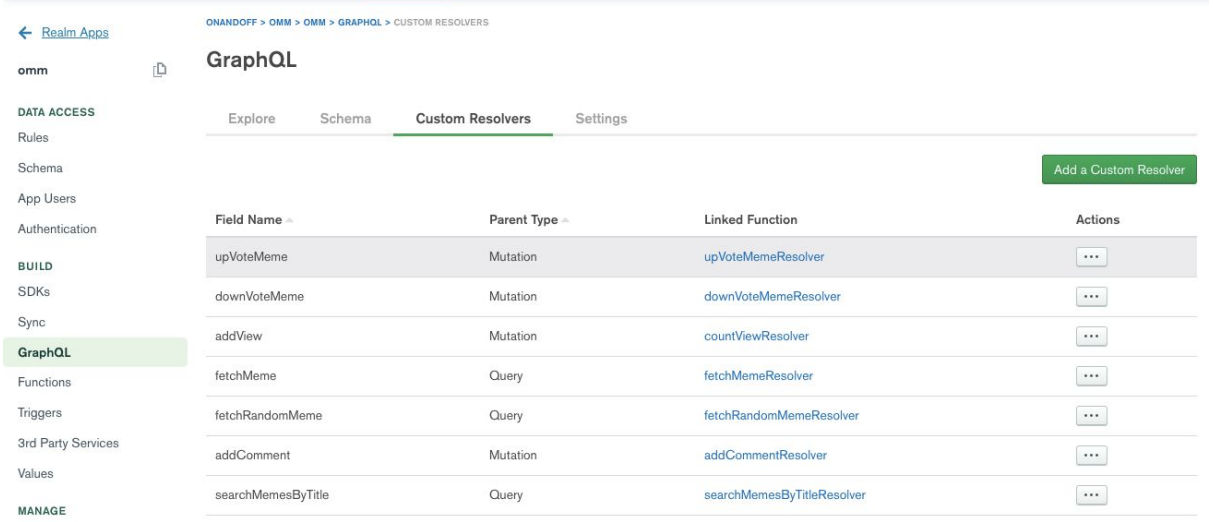

MONGODB-ATLAS
omm.users
Rules
Schema
Relationships
{}

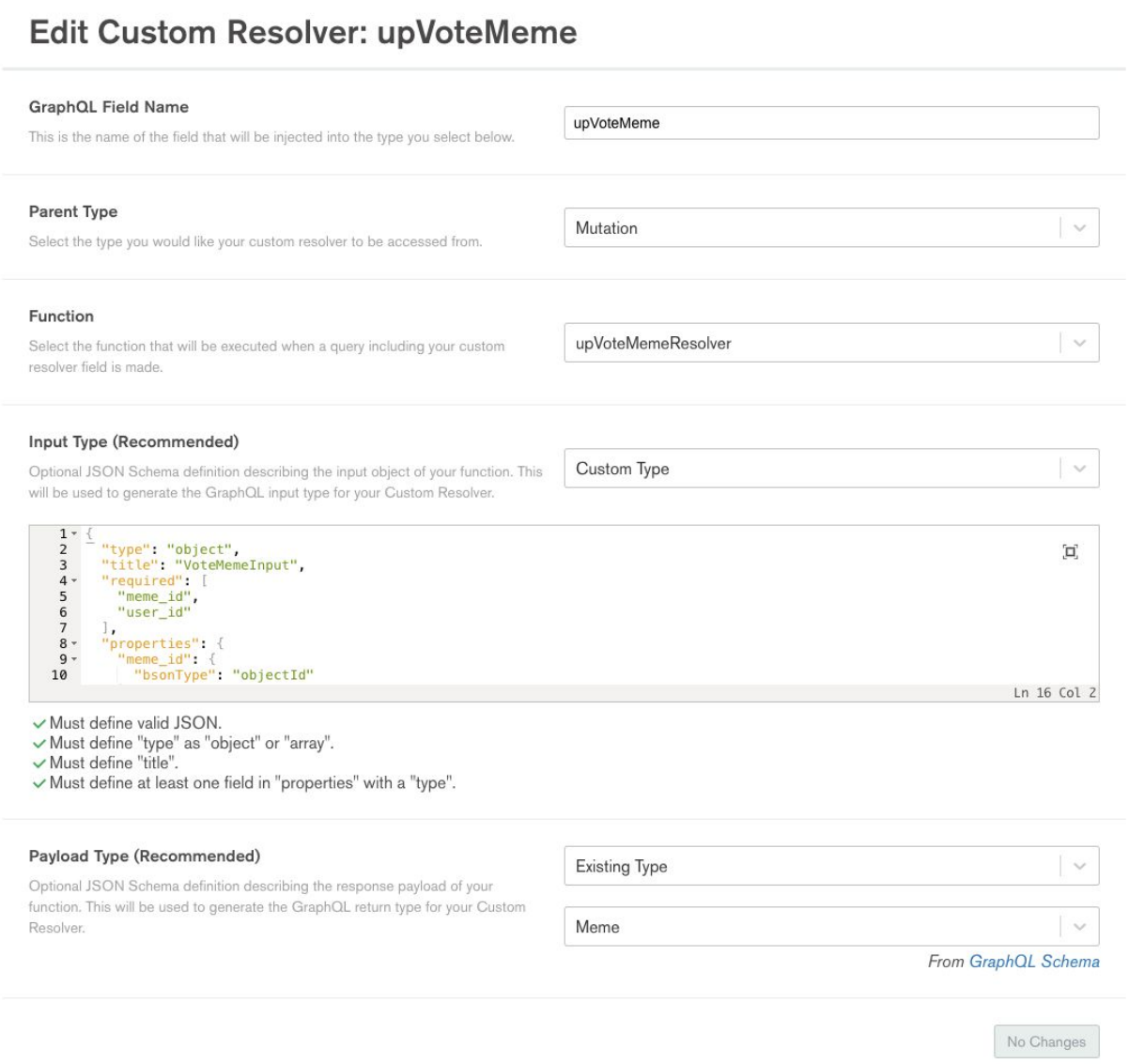
GraphQL

Get GraphQL URL here:



Custom Resolvers





InputType:

```
{
  "type": "object",
  "title": "VoteMemeInput",
  "required": [
    "meme_id",
    "user_id"
  ],
  "properties": {
    "meme_id": {
      "bsonType": "objectId"
    },
    "user_id": {
      "bsonType": "objectId"
    }
  }
}
```

Realm Apps

omm

DATA ACCESS
Rules
Schema
App Users
Authentication

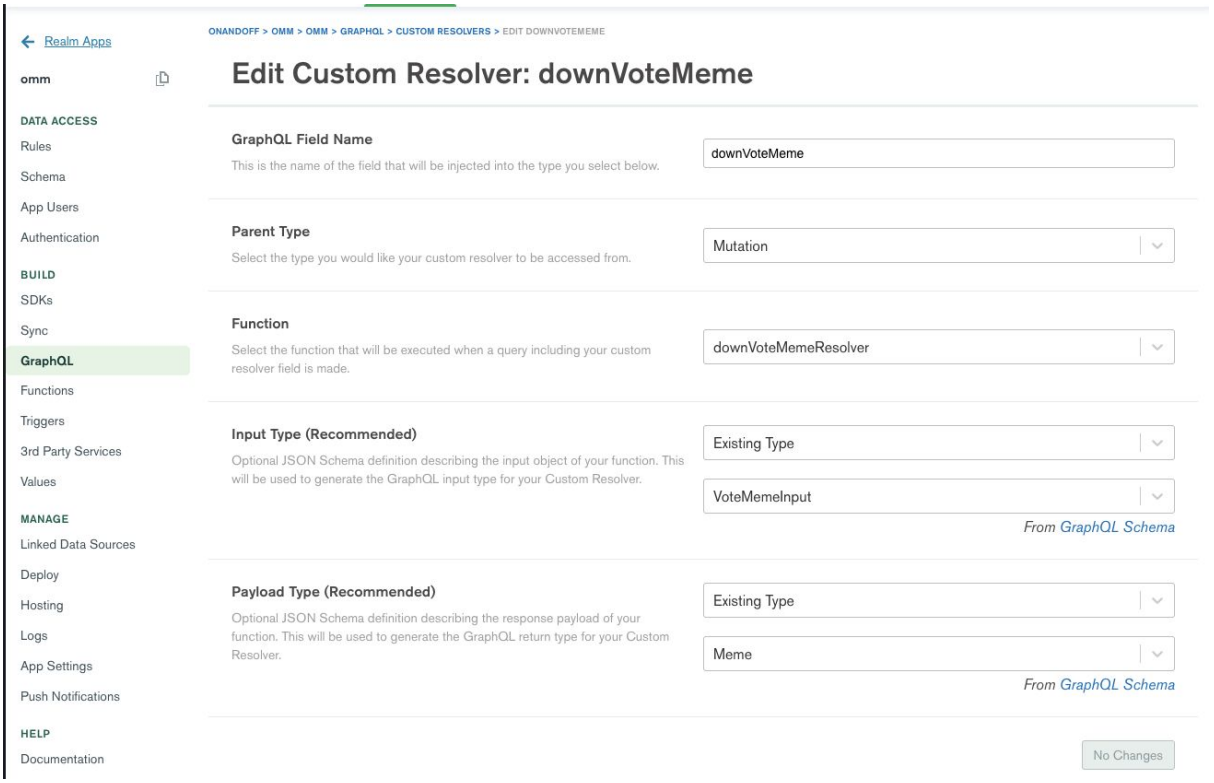
BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
Values

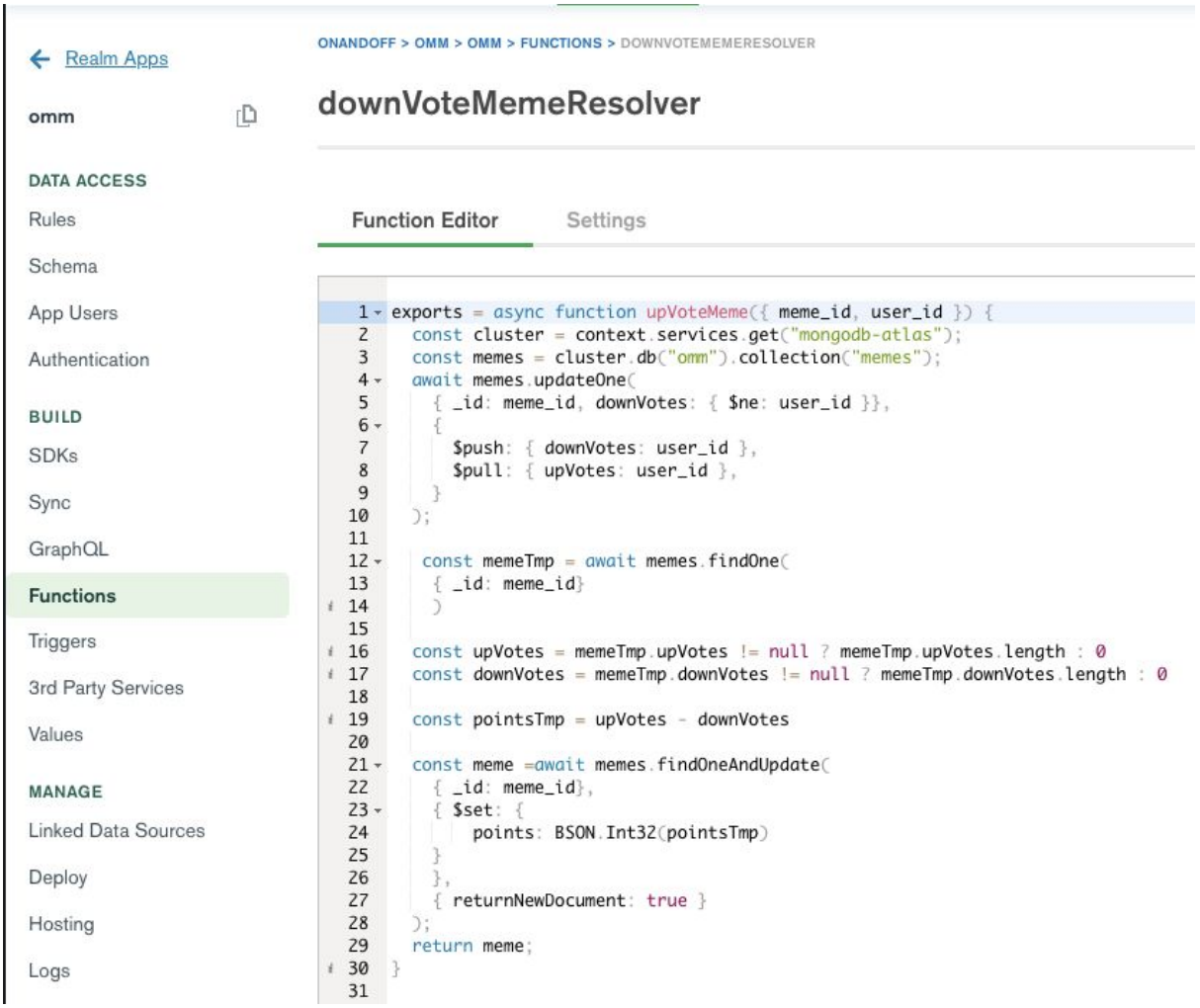
MANAGE
Linked Data Sources
Deploy
Hosting
Logs

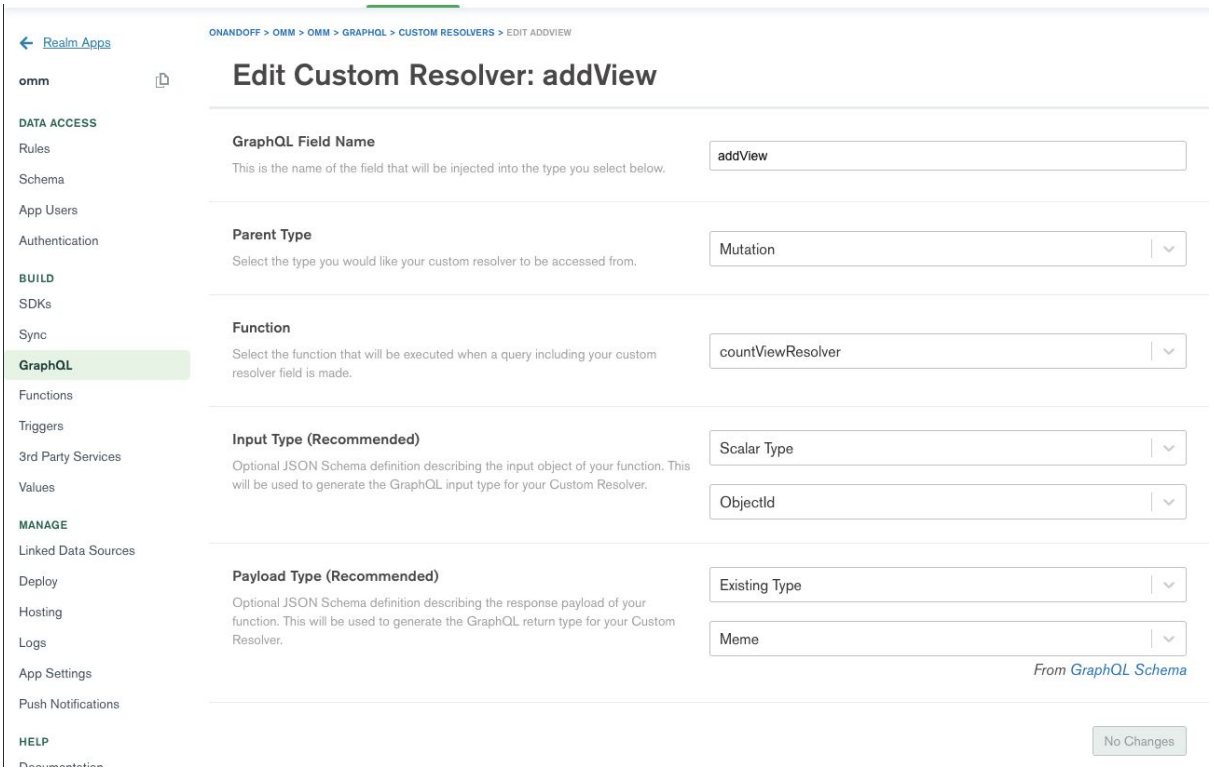
upVoteMemeResolver

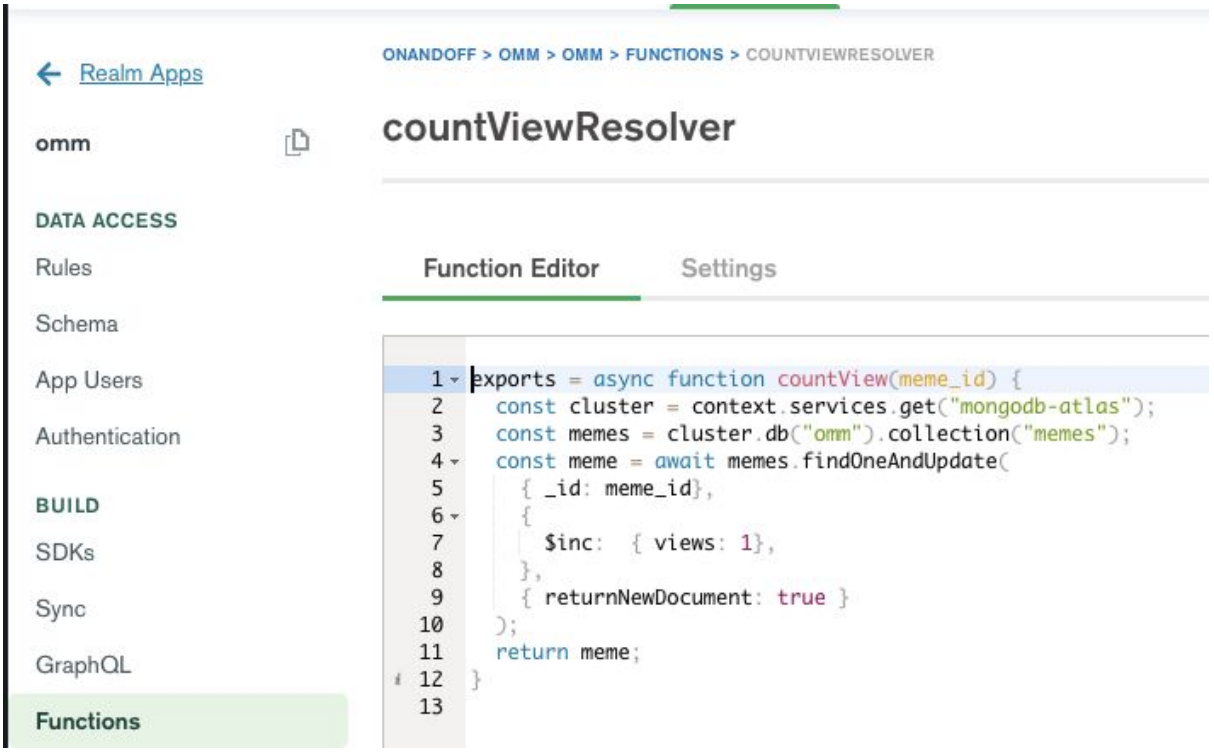
Function Editor | Settings

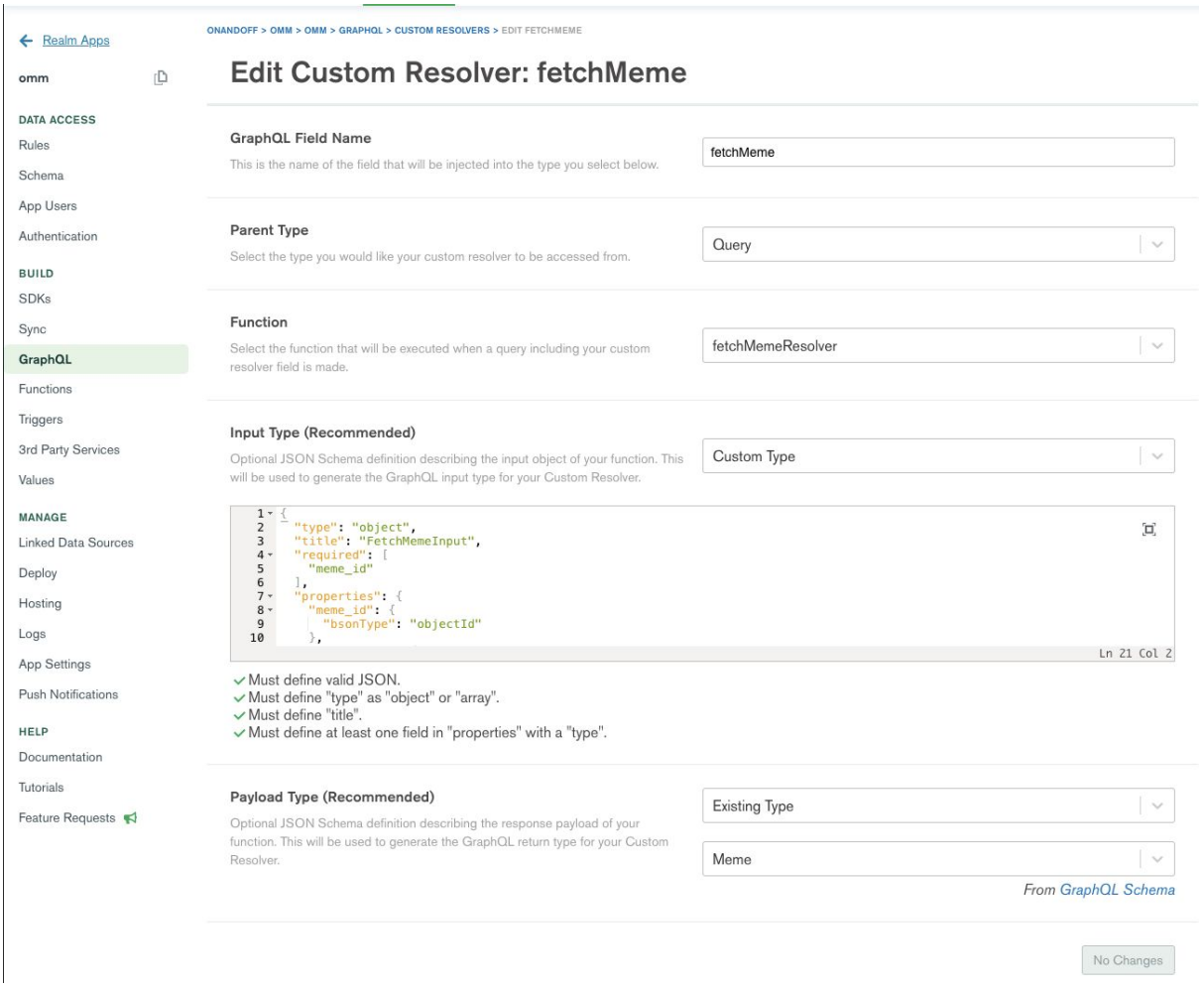
```
exports = async function upVoteMeme({ meme_id, user_id }) {
  const cluster = context.services.get("mongodb-atlas");
  const memes = cluster.db("omm").collection("memes");
  await memes.updateOne(
    { _id: meme_id, upVotes: { $ne: user_id }},
    { $push: { upVotes: user_id },
      $pull: { downVotes: user_id },
    }
  );

   const memeTmp = await memes.findOne(
    { _id: meme_id}
    )

  const upVotes = memeTmp.upVotes != null ? memeTmp.upVotes.length : 0
  const downVotes = memeTmp.downVotes != null ? memeTmp.downVotes.length : 0

  const pointsTmp = upVotes - downVotes

  const meme =await memes.findOneAndUpdate(
    { _id: meme_id},
    { $set: {
        points: BSON.Int32(pointsTmp)
    }
    },
    { returnNewDocument: true }
  );
  return meme;
}
```


Realm Apps
omm
DATA ACCESS
Rules
Schema
App Users
Authentication
BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
Values
MANAGE
Linked Data Sources
Deploy
Hosting
Logs
App Settings
Push Notifications
HELP
Documentation
ONANDOFF > OMM > OMM > GRAPHQL > CUSTOM RESOLVERS > EDIT DOWNVOTEMEME
Edit Custom Resolver: downVoteMeme
GraphQL Field Name
This is the name of the field that will be injected into the type you select below.
downVoteMeme
Parent Type
Select the type you would like your custom resolver to be accessed from.
Mutation
Function
Select the function that will be executed when a query including your custom resolver field is made.
downVoteMemeResolver
Input Type (Recommended)
Optional JSON Schema definition describing the input object of your function. This will be used to generate the GraphQL input type for your Custom Resolver.
Existing Type
VoteMemeInput
From GraphQL Schema
Payload Type (Recommended)
Optional JSON Schema definition describing the response payload of your function. This will be used to generate the GraphQL return type for your Custom Resolver.
Existing Type
Meme
From GraphQL Schema
No Changes


Realm Apps
omm
DATA ACCESS
Rules
Schema
App Users
Authentication
BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
Values
MANAGE
Linked Data Sources
Deploy
Hosting
Logs
ONANDOFF > OMM > OMM > FUNCTIONS > DOWNVOTEMEMERESOLVER
downVoteMemeResolver
Function Editor
Settings
exports = async function upVoteMeme({ meme_id, user_id }) {
const cluster = context.services.get("mongodb-atlas");
const memes = cluster.db("omm").collection("memes");
await memes.updateOne(
{ _id: meme_id, downVotes: { $ne: user_id }},
{
$push: { downVotes: user_id },
$pull: { upVotes: user_id },
}
);
const memeTmp = await memes.findOne(
{ _id: meme_id}
)
const upVotes = memeTmp.upVotes != null ? memeTmp.upVotes.length : 0
const downVotes = memeTmp.downVotes != null ? memeTmp.downVotes.length : 0
const pointsTmp = upVotes - downVotes
const meme =await memes.findOneAndUpdate(
{ _id: meme_id},
{ $set: {
points: BSON.Int32(pointsTmp)
}
},
{ returnNewDocument: true }
);
return meme;
}


Realm Apps
omm
DATA ACCESS
Rules
Schema
App Users
Authentication
BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
Values
MANAGE
Linked Data Sources
Deploy
Hosting
Logs
App Settings
Push Notifications
HELP
ONANDOFF > OMM > OMM > GRAPHQL > CUSTOM RESOLVERS > EDIT ADDVIEW
Edit Custom Resolver: addView
GraphQL Field Name
This is the name of the field that will be injected into the type you select below.
addView
Parent Type
Select the type you would like your custom resolver to be accessed from.
Mutation
Function
Select the function that will be executed when a query including your custom resolver field is made.
countViewResolver
Input Type (Recommended)
Optional JSON Schema definition describing the input object of your function. This will be used to generate the GraphQL input type for your Custom Resolver.
Scalar Type
ObjectId
Payload Type (Recommended)
Optional JSON Schema definition describing the response payload of your function. This will be used to generate the GraphQL return type for your Custom Resolver.
Existing Type
Meme
From GraphQL Schema
No Changes


Realm Apps
omm
DATA ACCESS
Rules
Schema
App Users
Authentication
BUILD
SDKs
Sync
GraphQL
Functions
ONANDOFF > OMM > OMM > FUNCTIONS > COUNTVIEWRESOLVER
countViewResolver
Function Editor
Settings
exports = async function countView(meme_id) {
const cluster = context.services.get("mongodb-atlas");
const memes = cluster.db("omm").collection("memes");
const meme = await memes.findOneAndUpdate(
{ _id: meme_id},
{
$inc: { views: 1},
},
{ returnNewDocument: true }
);
return meme;
}



Input Type:

```
{
  "type": "object",
  "title": "FetchMemeInput",
  "required": [
    "meme_id"
  ],
  "properties": {
    "meme_id": {
      "bsonType": "objectId"
    },
    "conditions": {
      "bsonType": "string"
    },
    "sorts": {
      "bsonType": "string"
    },
    "next": {
      "bsonType": "boolean"
    }
  }
}
```

Realm Apps

omm

DATA ACCESS
Rules
Schema
App Users
Authentication

BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
Values

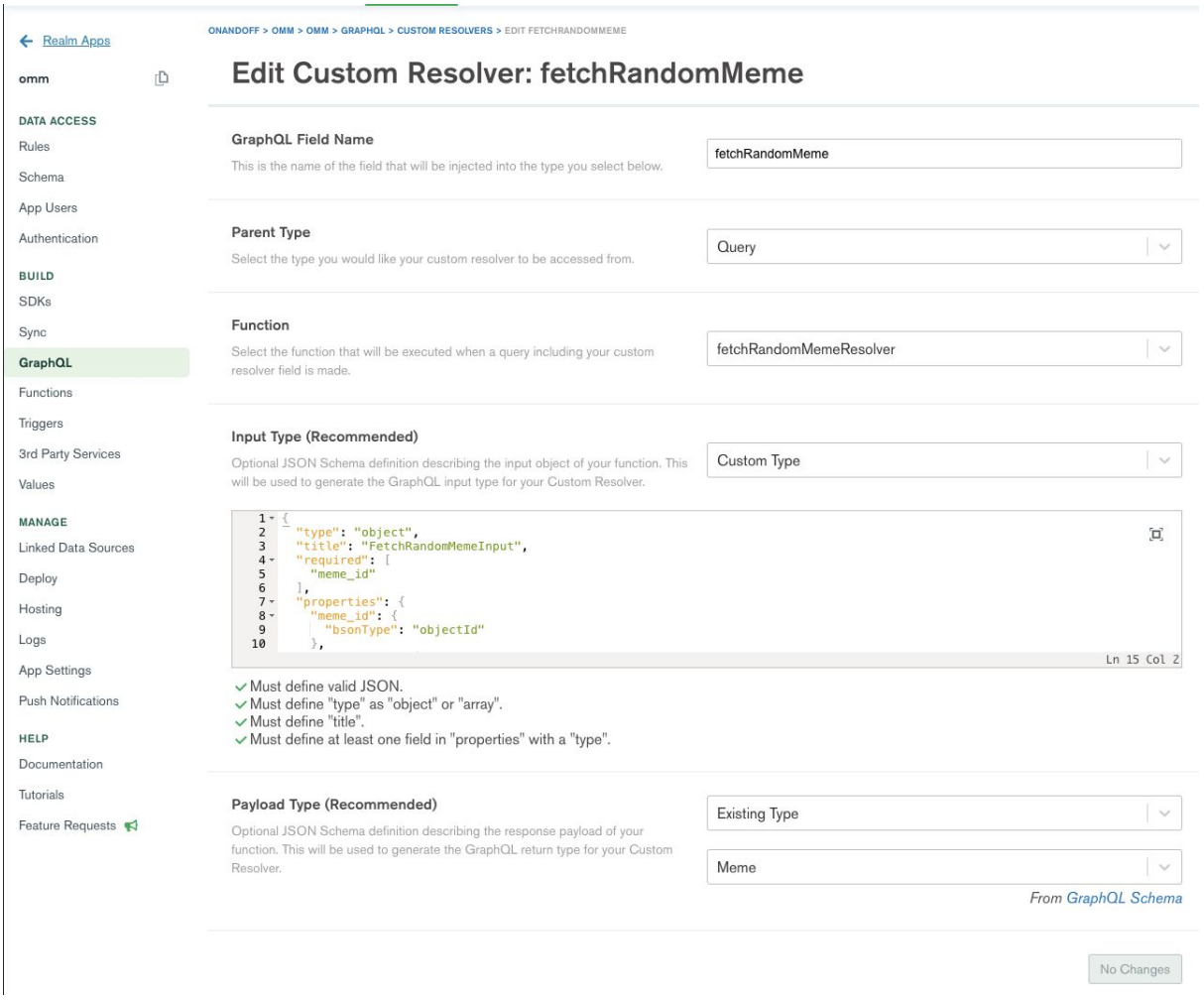
MANAGE
Linked Data Sources
Deploy
Hosting
Logs
App Settings
Push Notifications

HELP
Documentation
Tutorials

fetchMemeResolver

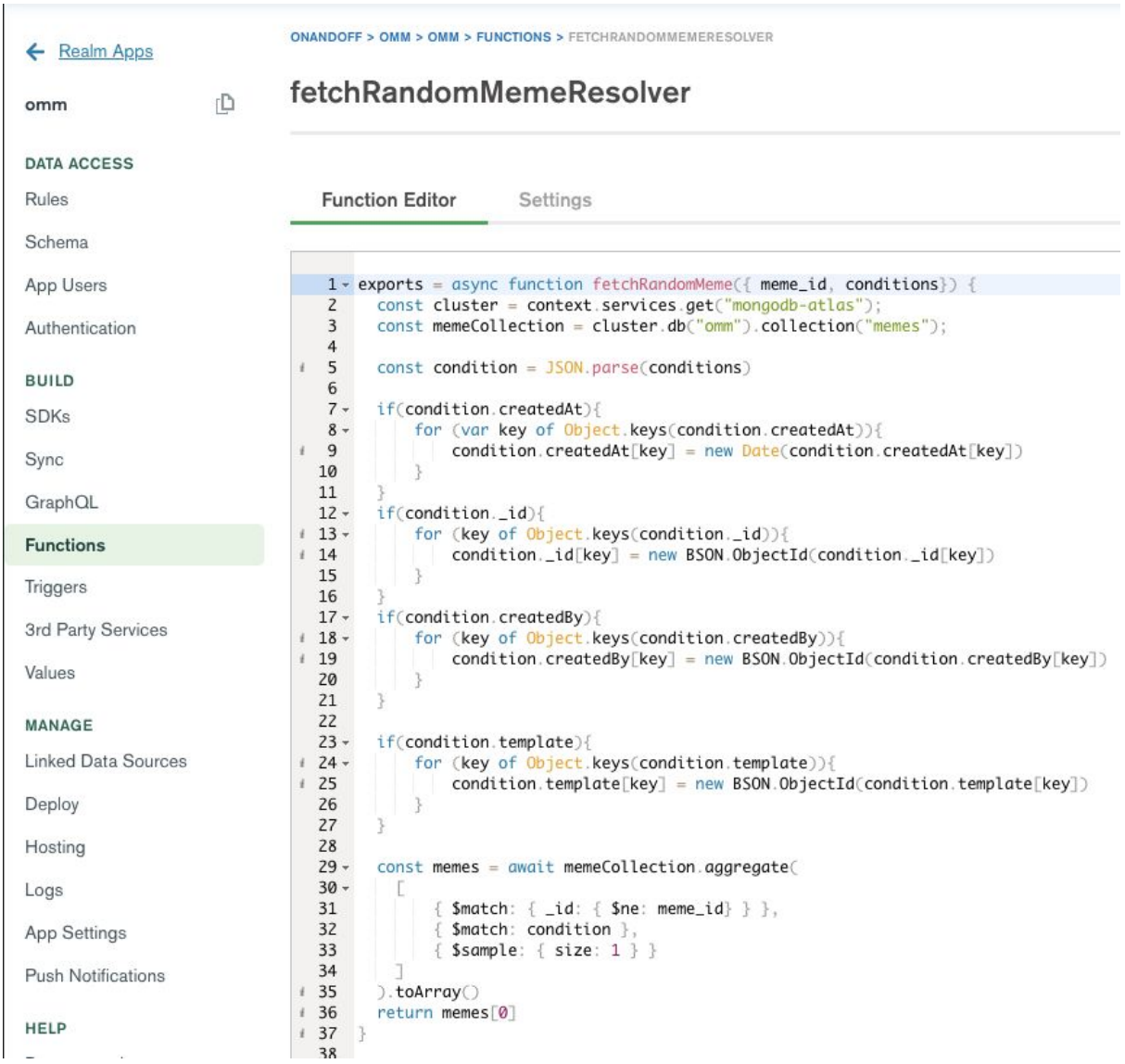
Function Editor | Settings

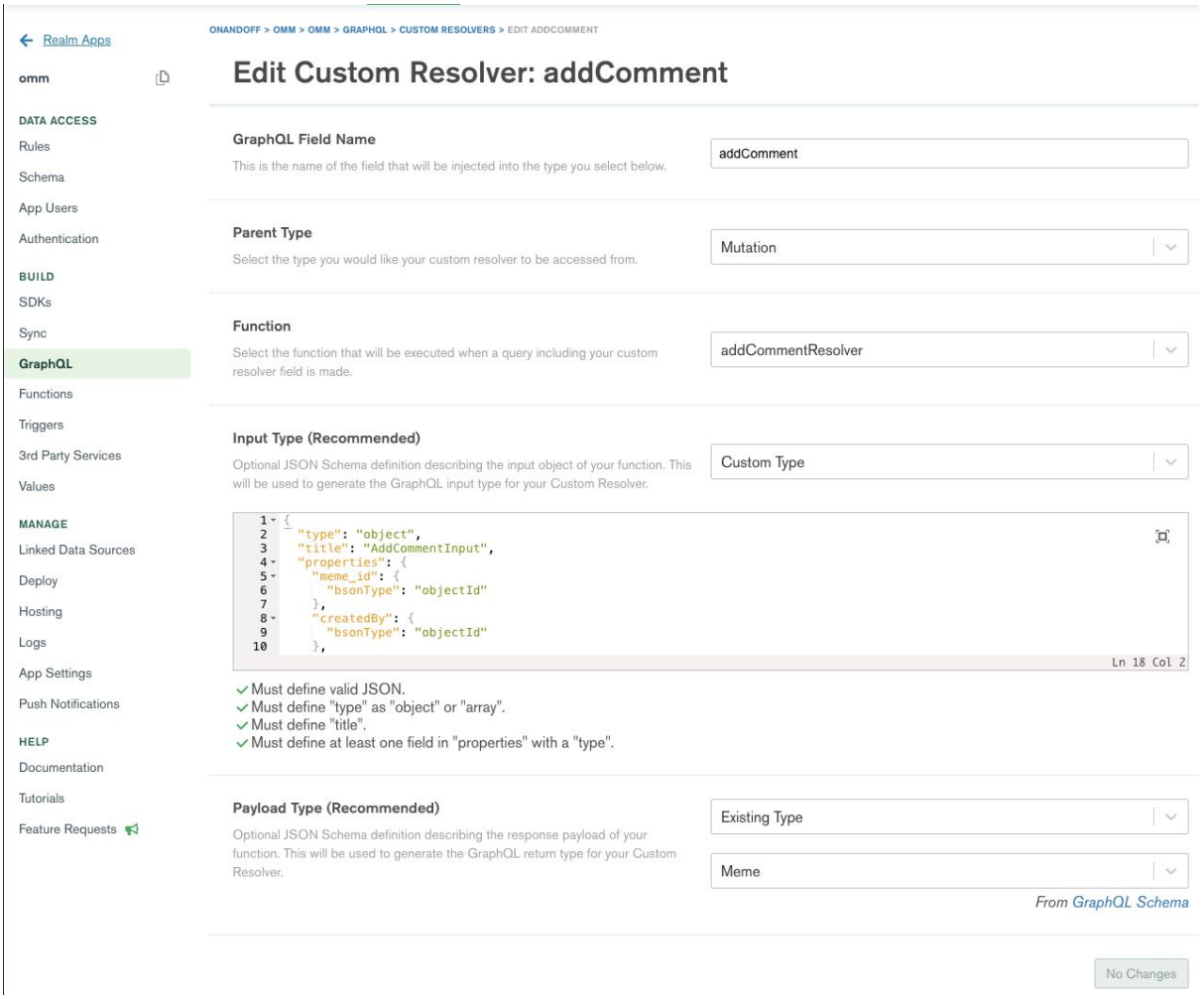
```
exports = async function fetchMeme({ meme_id, conditions, sorts, next }) {

  const cluster = context.services.get("mongodb-atlas");
  const memeCollection = cluster.db("omm").collection("memes");

  const sort = JSON.parse(sorts)
  const condition = JSON.parse(conditions)

  if(condition.createdAt){
      for (var key of Object.keys(condition.createdAt)){
          condition.createdAt[key] = new Date(condition.createdAt[key])
      }
  }
  if(condition._id){
      for (key of Object.keys(condition._id)){
          condition._id[key] = new BSON.ObjectId(condition._id[key])
      }
  }
  if(condition.createdBy){
      for (key of Object.keys(condition.createdBy)){
          condition.createdBy[key] = new BSON.ObjectId(condition.createdBy[key])
      }
  }

  if(condition.template){
      for (key of Object.keys(condition.template)){
          condition.template[key] = new BSON.ObjectId(condition.template[key])
      }
  }

  const memes = await memeCollection.find(condition).sort(sort).toArray()

  const index = memes.findIndex(x => x._id.toString() === meme_id.toString())

  if(next){
    return index + 1 >= memes.length ? null : memes[index + 1]
  } else{
    return index - 1 < 0 ? null : memes[index - 1]
  }

}
```



Input Type:

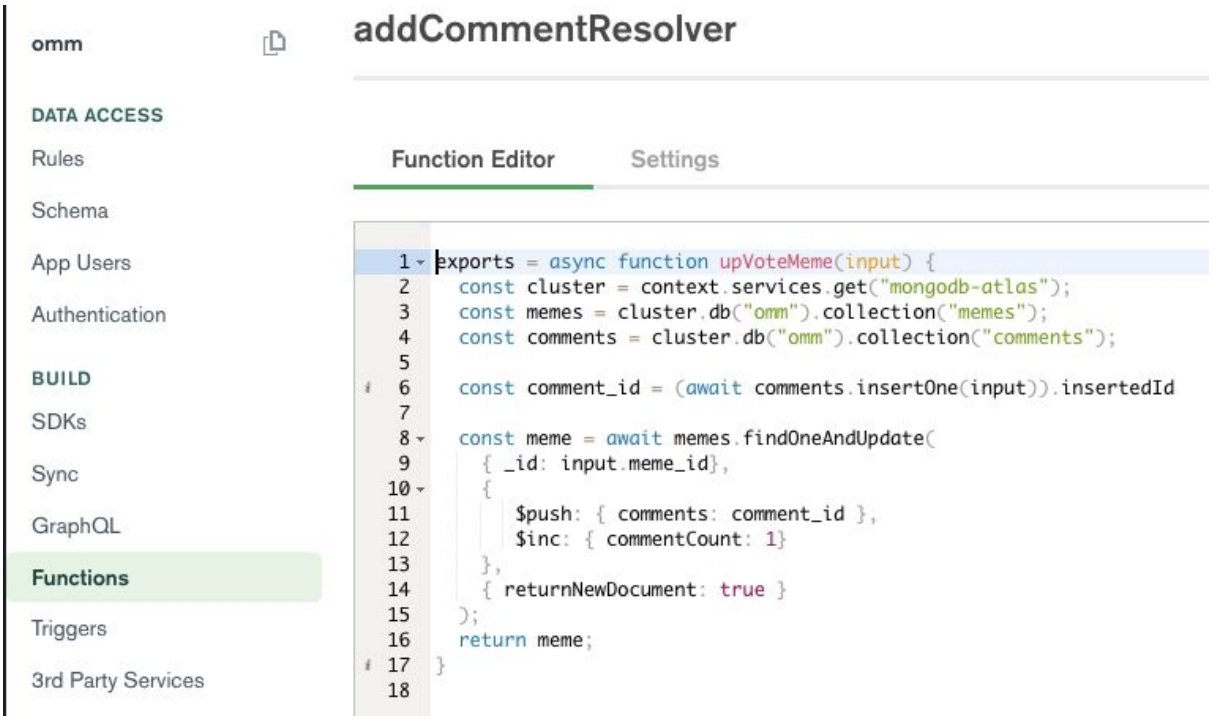
```
{
  "type": "object",
  "title": "FetchRandomMemeInput",
  "required": [
    "meme_id"
  ],
  "properties": {
    "meme_id": {
      "bsonType": "objectId"
    },
    "conditions": {
      "bsonType": "string"
    }
  }
}
```

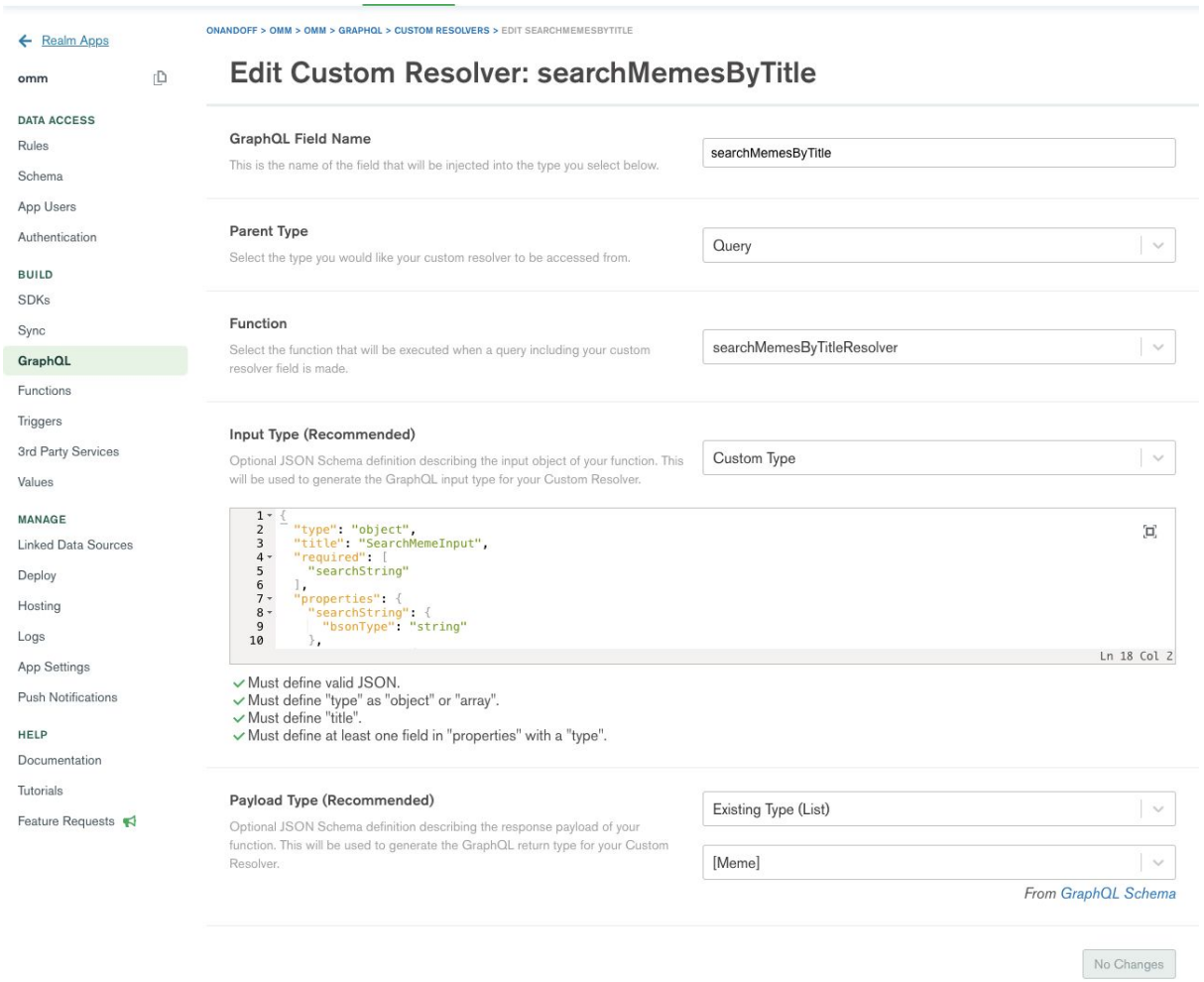

Realm Apps
omm
DATA ACCESS
Rules
Schema
App Users
Authentication
BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
Values
MANAGE
Linked Data Sources
Deploy
Hosting
Logs
App Settings
Push Notifications
HELP
ONANDOFF > OMM > OMM > FUNCTIONS > FETCHRANDOMMEMERESOLVER
fetchRandomMemeResolver
Function Editor
Settings
exports = async function fetchRandomMeme({ meme_id, conditions}) {
const cluster = context.services.get("mongodb-atlas");
const memeCollection = cluster.db("omm").collection("memes");
const condition = JSON.parse(conditions)
if(condition.createdAt){
for (var key of Object.keys(condition.createdAt)){
condition.createdAt[key] = new Date(condition.createdAt[key])
}
}
if(condition._id){
for (key of Object.keys(condition._id)){
condition._id[key] = new BSON.ObjectId(condition._id[key])
}
}
if(condition.createdBy){
for (key of Object.keys(condition.createdBy)){
condition.createdBy[key] = new BSON.ObjectId(condition.createdBy[key])
}
}
if(condition.template){
for (key of Object.keys(condition.template)){
condition.template[key] = new BSON.ObjectId(condition.template[key])
}
}
const memes = await memeCollection.aggregate(
[
{ $match: { _id: { $ne: meme_id} } },
{ $match: condition },
{ $sample: { size: 1 } }
]
).toArray()
return memes[0]
}



Input Type:

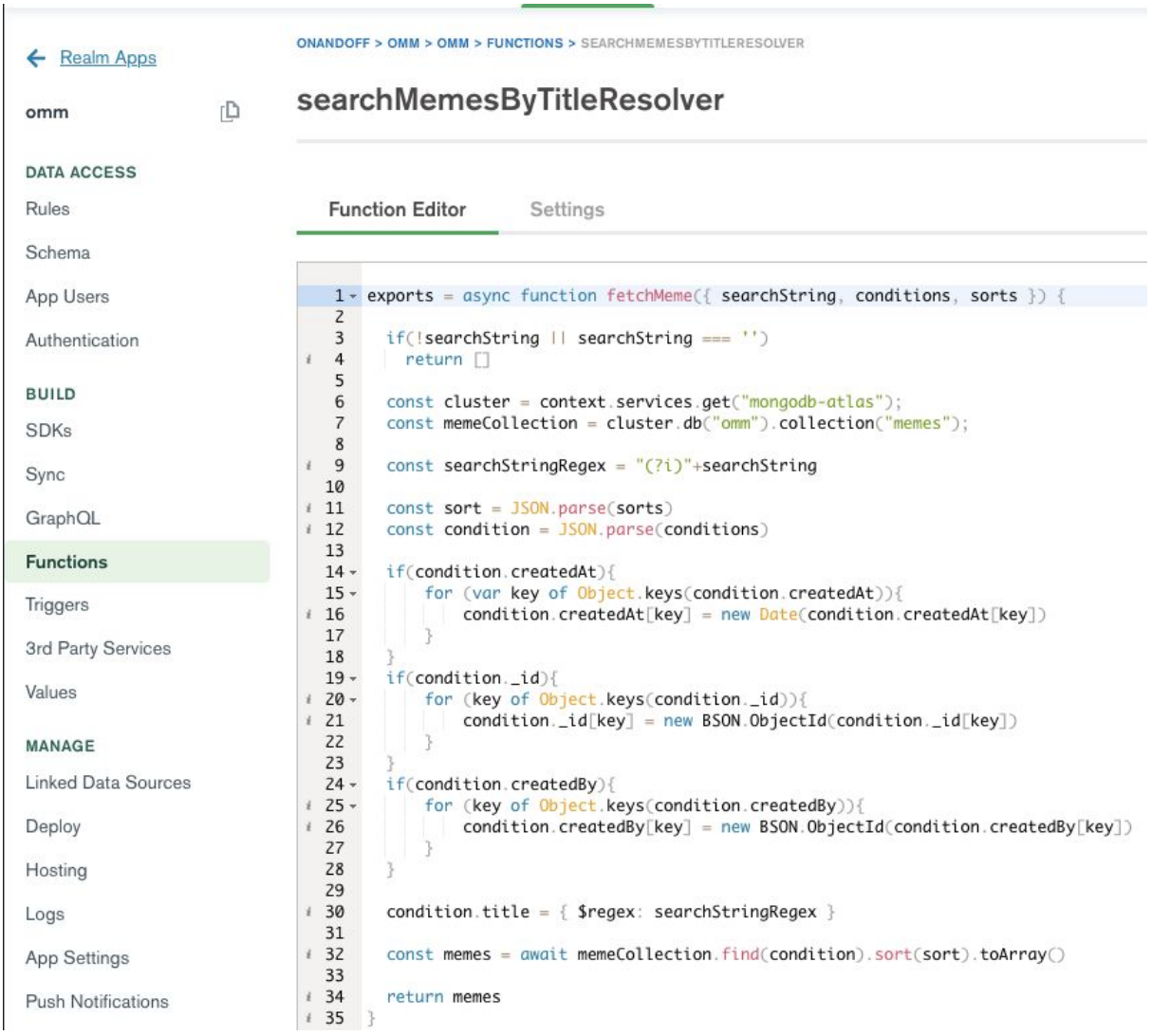
```
{
  "type": "object",
  "title": "AddCommentInput",
  "properties": {
    "meme_id": {
      "bsonType": "objectId"
    },
    "createdBy": {
      "bsonType": "objectId"
    },
    "text": {
      "bsonType": "string"
    },
    "createdAt": {
      "bsonType": "date"
    }
  }
}
```

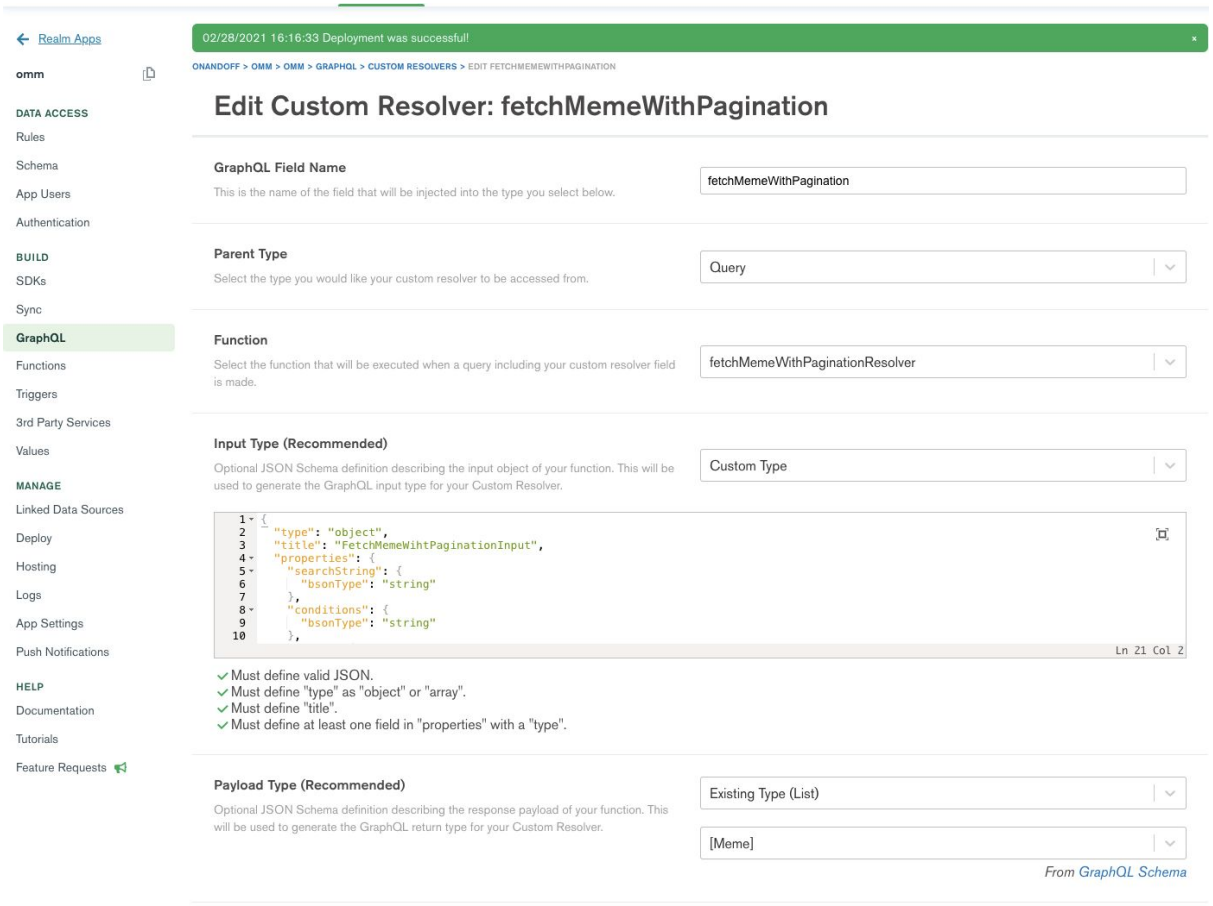

omm
DATA ACCESS
Rules
Schema
App Users
Authentication
BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
addCommentResolver
Function Editor
Settings
exports = async function upVoteMeme(input) {
const cluster = context.services.get("mongodb-atlas");
const memes = cluster.db("omm").collection("memes");
const comments = cluster.db("omm").collection("comments");
const comment_id = (await comments.insertOne(input)).insertedId
const meme = await memes.findOneAndUpdate(
{ _id: input.meme_id},
{
$push: { comments: comment_id },
$inc: { commentCount: 1}
},
{ returnNewDocument: true }
);
return meme;
}



Input Type:

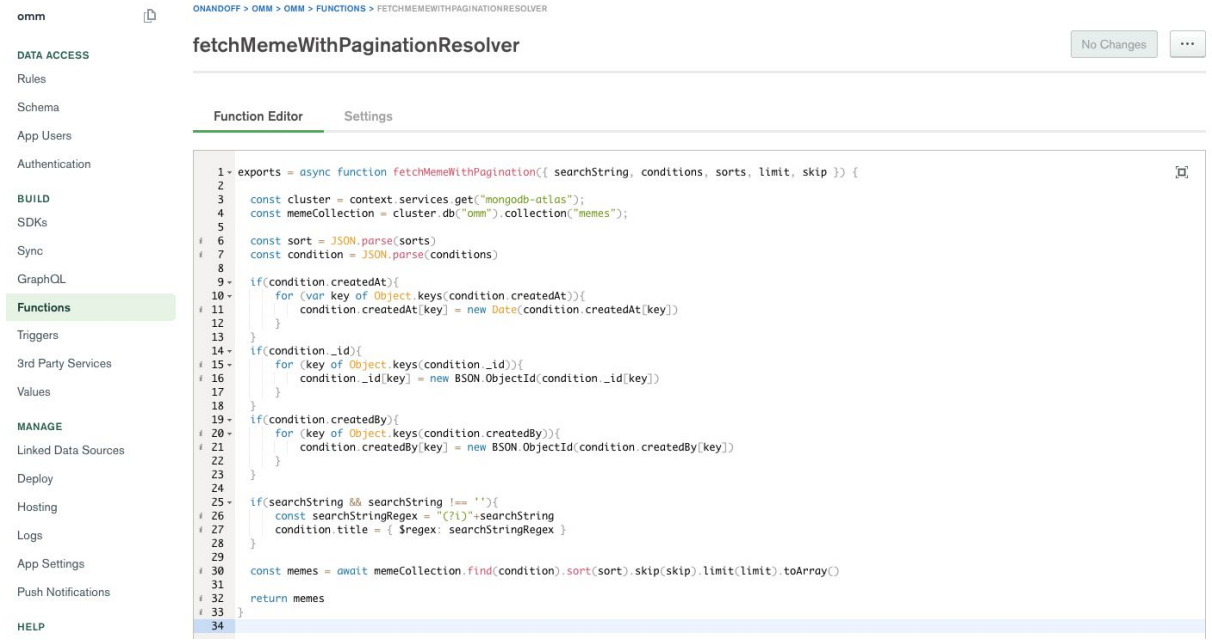
```
{
  "type": "object",
  "title": "SearchMemeInput",
  "required": [
    "searchString"
  ],
  "properties": {
    "searchString": {
      "bsonType": "string"
    },
    "conditions": {
      "bsonType": "string"
    },
    "sorts": {
      "bsonType": "string"
    }
  }
}
```


Realm Apps
omm
DATA ACCESS
Rules
Schema
App Users
Authentication
BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
Values
MANAGE
Linked Data Sources
Deploy
Hosting
Logs
App Settings
Push Notifications
ONANDOFF > OMM > OMM > FUNCTIONS > SEARCHMEMESBYTITLERESOLVER
searchMemesByTitleResolver
Function Editor
Settings
exports = async function fetchMeme({ searchString, conditions, sorts }) {
if(!searchString || searchString === '')
return []
const cluster = context.services.get("mongodb-atlas");
const memeCollection = cluster.db("omm").collection("memes");
const searchStringRegex = "(?i)"+searchString
const sort = JSON.parse(sorts)
const condition = JSON.parse(conditions)
if(condition.createdAt){
for (var key of Object.keys(condition.createdAt)){
condition.createdAt[key] = new Date(condition.createdAt[key])
}
}
if(condition._id){
for (key of Object.keys(condition._id)){
condition._id[key] = new BSON.ObjectId(condition._id[key])
}
}
if(condition.createdBy){
for (key of Object.keys(condition.createdBy)){
condition.createdBy[key] = new BSON.ObjectId(condition.createdBy[key])
}
}
condition.title = { $regex: searchStringRegex }
const memes = await memeCollection.find(condition).sort(sort).toArray()
return memes
}



Input Type:

```
{
  "type": "object",
  "title": "FetchMemeWihtPaginationInput",
  "properties": {
    "searchString": {
      "bsonType": "string"
    },
    "conditions": {
      "bsonType": "string"
    },
    "sorts": {
      "bsonType": "string"
    },
    "limit": {
      "bsonType": "int"
    },
    "skip": {
      "bsonType": "int"
    }
  }
}
```


omm
DATA ACCESS
Rules
Schema
App Users
Authentication
BUILD
SDKs
Sync
GraphQL
Functions
Triggers
3rd Party Services
Values
MANAGE
Linked Data Sources
Deploy
Hosting
Logs
App Settings
Push Notifications
HELP
ONANDOFF > OMM > OMM > FUNCTIONS > FETCHMEMEWITHPAGINATIONRESOLVER
fetchMemeWithPaginationResolver
No Changes
Function Editor
Settings
exports = async function fetchMemeWithPagination({ searchString, conditions, sorts, limit, skip }) {
const cluster = context.services.get("mongodb-atlas");
const memeCollection = cluster.db("omm").collection("memes");
const sort = JSON.parse(sorts)
const condition = JSON.parse(conditions)
if(condition.createdAt){
for (var key of Object.keys(condition.createdAt)){
condition.createdAt[key] = new Date(condition.createdAt[key])
}
}
if(condition._id){
for (key of Object.keys(condition._id)){
condition._id[key] = new BSON.ObjectId(condition._id[key])
}
}
if(condition.createdBy){
for (key of Object.keys(condition.createdBy)){
condition.createdBy[key] = new BSON.ObjectId(condition.createdBy[key])
}
}
if(searchString && searchString !== ''){
const searchStringRegex = "(?i)"+searchString
condition.title = { $regex: searchStringRegex }
}
const memes = await memeCollection.find(condition).sort(sort).skip(skip).limit(limit).toArray()
return memes
}

Screenshot API - ApiFlash

ACCESS

The API can be accessed through a unique HTTPS endpoint that supports both GET and POST methods.

GET https://api.apiflash.com/v1/urltoimage
└ Parameters are passed through **query string**.

POST https://api.apiflash.com/v1/urltoimage
└ Parameters are passed as **form data**.

All API calls needs to be authenticated using a valid access key that can be found in the dashboard. For a GET request, the access key is passed in the query string as all other parameters.

GET https://api.apiflash.com/v1/urltoimage?access_key=**YOUR_ACCESS_KEY**

For a POST request, the access key is passed as form data as every other parameter.

By default the API returns directly the screenshot image data. It also passes along the appropriate `Content-Type` and `Content-Length` headers. If the `response_type` parameter is set to `json`, then the response contains a JSON document with links to the resulting screenshot.

EXAMPLES

The following examples make a HTTP request to the API and save the screenshot to a `screenshot.jpeg` file.

C# CURL GO JAVA NODE PHP PYTHON RUBY

```
const request = require('request');
const fs = require('fs');

request({
    url: "https://api.apiflash.com/v1/urltoimage",
    encoding: "binary",
    qs: {
        access_key: "YOUR_ACCESS_KEY",
        url: "https://example.com"
    }
}, (error, response, body) => {
    if (error) {
        console.log(error);
    } else {
        fs.writeFile("screenshot.jpeg", body, "binary", error => {
            console.log(error);
        });
    }
});
```

Free Plan

$ **0** / month

- ✓ **100 Screenshots**
- ✓ Full page screenshots
- ✓ Mobile screenshots
- ✓ Basic support

Your current plan

set up account to access API key