

Firestore	1
MongoDB Atlas & Realm	4
Screenshot API - ApiFlash	27

## Firestore

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

In order to connect firestore 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:

The screenshot displays the Firebase console interface. On the left, a dark sidebar contains navigation links for 'Entwickeln' (Authentication, Cloud Firestore, Realtime Database, Storage, Hosting, Functions, Machine Learning) and 'Veröffentlichen und beobachten' (Crashlytics, Performance, Test Lab). The main content area is titled 'Projekteinstellungen' (Project Settings) for the project 'online-multimedia'. It features tabs for 'Allgemein', 'Cloud Messaging', 'Integrationen', 'Dienstkonten', 'Datenschutz', and 'Nutzer und Berechtigungen'. The 'Allgemein' tab is active, showing project details: Projectname (online-multimedia), Projekt-ID (online-multimedia), Projektnummer (593943106717), Standardmäßiger GCP-Ressourcenstandort (eur3 (eu-central-1)), and Web-API-Schlüssel (AlzaSyBu9gRj4xOHPJl8XQWYZX5M3wq\_tU3mPBQ). Below this, the 'Öffentliche Einstellungen' section shows the Öffentlicher Name (project-593943106717) and Support-E-Mail-Adresse (Nicht konfiguriert). A 'Meine Apps' section lists a 'Web-App' named 'OMM-Vercel' with its App-ID (1:593943106717:web:7b7b263a79d112e3d7074) and a link to the Firebase Hosting page. At the bottom, the 'Firestore SDK snippet' section shows the 'CDN' option selected, with instructions to copy and paste the provided script tags into the app's HTML body.

## Admin configuration:

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

The screenshot shows the 'Projekteinstellungen' (Project Settings) page for 'online-multimedia'. The 'Dienstkonten' (Service Accounts) tab is selected. On the left, there's a sidebar with navigation options like 'Entwickeln' (Develop) and 'Analytics'. The main content area shows the 'Firebase Admin SDK' configuration. It includes a section for 'Alte Anmeldedaten' (Old login data) and 'Datenbank-Secrets' (Database secrets). Below that, it lists 'Andere Dienstkonten' (Other service accounts) with '4 Dienstkonten von Google Cloud'. The right side of the page displays the 'Firebase Admin SDK' configuration details, including the 'Firebase-Dienstkonto' (firebase-adminsdk-f93wu@online-multimedia.iam.gserviceaccount.com) and a code snippet for the Admin SDK configuration. At the bottom, there's a button labeled 'Neuen privaten Schlüssel generieren' (Generate new private key).

**Projekteinstellungen**

Allgemein Cloud Messaging Integrationen **Dienstkonten** Datenschutz Nutzer und Berechtigungen

[Dienstkontoberechtigungen verwalten](#)

**Firebase Admin SDK**

Ihr Firebase-Dienstkonto kann verwendet werden, um verschiedene Firebase-Funktionen wie "Database", "Storage" und "Auth" programmatisch über das einheitliche Admin SDK zu authentifizieren. [Weitere Informationen](#)

Firebase-Dienstkonto  
firebase-adminsdk-f93wu@online-multimedia.iam.gserviceaccount.com

Snippet für Admin SDK-Konfiguration

☒ Node.js ☐ Java ☐ Python ☐ Go

```
var admin = require("firebase-admin");  
  
var serviceAccount = require("path/to/serviceAccountKey.json");  
  
admin.initializeApp({  
  credential: admin.credential.cert(serviceAccount),  
  databaseURL: "https://online-multimedia.firebaseio.com"  
});
```

[Neuen privaten Schlüssel generieren](#)

## Authentication Settings (->Activate “anonym”)

The screenshot shows the 'Authentication' settings page in the Firebase console. The 'Sign-in method' tab is selected. The page displays a list of 'Anbieter für Anmeldungen' (Providers for sign-in). The 'Anonym' provider is listed at the bottom and is currently 'Aktiviert' (Activated). All other providers are 'Deaktiviert' (Deactivated).

**Authentication**

Users **Sign-in method** Templates Usage

Anbieter für Anmeldungen

Anbieter	Status
E-Mail-Adresse/Passwort	Deaktiviert
Telefon	Deaktiviert
Google	Deaktiviert
Play Spiele	Deaktiviert
Game Center	Deaktiviert
Facebook	Deaktiviert
Twitter	Deaktiviert
GitHub	Deaktiviert
Yahoo!	Deaktiviert
Microsoft	Deaktiviert
Apple	Deaktiviert
Anonym	Aktiviert

# Cloud Firestore Rules

The screenshot shows the Firebase Cloud Firestore Rules editor. The left sidebar contains the Firebase logo and a navigation menu with 'Projektübersicht' and 'Entwickeln'. The 'Entwickeln' section lists services: Authentication, Cloud Firestore (selected), Realtime Database, Storage, Hosting, Functions, and Machine Learning. The main area is titled 'Cloud Firestore' and shows the project name 'online-multimedia'. It has tabs for 'Daten', 'Regeln' (selected), 'Indexe', and 'Nutzung'. Below the tabs are buttons for 'Regeln bearbeiten' and 'Regeln überwachen', and a blue 'Entwickeln und Testen' button. A timeline on the left shows three events: 'Heute • 2:10 PM' (with a star icon), 'Dez. 26, 2020 • 2:41 PM', and 'Dez. 26, 2020 • 2:38 PM'. The right pane displays the current rule code in a text editor with line numbers 1 through 7.

```
1 service cloud.firestore {  
2   match /databases/{database}/documents {  
3     match /{document=**} {  
4       allow read, write: if request.auth != null || request.auth == null  
5     }  
6   }  
7 }
```

# 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**:

The screenshot shows the MongoDB Atlas interface for a cluster named 'omm-cluster0'. The left sidebar contains navigation links: DATA STORAGE (Clusters, Triggers, Data Lake), SECURITY (Database Access, Network Access, Advanced), and a search bar. The main content area displays cluster details: 'omm-cluster0' (Version 4.4.4), 'CONNECT', 'METRICS', and 'COLLECTIONS' buttons. Below these are fields for CLUSTER TIER (M0 Sandbox (General)), REGION (AWS / Frankfurt (eu-central-1)), TYPE (Replica Set - 3 nodes), and LINKED REALM APP (omm). To the right, there are two graphs: 'Operations' (Last 6 Hours) showing a peak of 3.9/s, and 'Connections' (Last 6 Hours) showing a peak of 500. A 'Logical Size' bar chart shows 232.1 KB out of 512.0 MB max. A 'Create a New Cluster' button is in the top right. A banner at the top mentions Multi Factor Authentication.

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

The screenshot shows the 'Connect to omm-cluster0' dialog. It has a progress bar with three steps: 'Setup connection security' (checked), 'Choose a connection method' (checked), and 'Connect'. Step 1, 'Select your driver and version', shows 'DRIVER' set to 'PHP' and 'VERSION' set to 'PHPLIB 1.8'. Step 2, 'Add your connection string into your application code', includes a checkbox for 'Include full driver code example' and a text box containing the connection string: `mongodb+srv://omm-mongodb:<password>@omm-cluster0.f8ptd.mongodb.net/myFirstDatabase?retryWrites=true&w=majority`. Below the text box, instructions state: 'Replace <password> with the password for the omm-mongoDB user. Replace myFirstDatabase with the name of the database that connections will use by default. Ensure any option params are URL encoded.' A link for 'View our troubleshooting documentation' is provided. At the bottom are 'Go Back' and 'Close' buttons.

Get NEXT\_PUBLIC\_MONGODB\_REALM\_ID here (App ID):

omm

Atlas

Realm

Charts

← Realm Apps

omm

App ID: omm-edgws

Guides

DATA ACCESS

Rules

Schema

App Users

Authentication

BUILD

SDKs

Sync

GraphQL

Functions

Triggers

3rd Party Services

Values

You're almost done setting up your application...

✓ Link an Atlas Data Source

Securely query and work with your data in your linked Atlas cluster or Data Lake.

✓ omm-cluster0 Linked

✓ Add a Collection

Click to create a new collection or add an existing one from your Atlas cluster.

✓ Completed

3 Set Data Access Rules

Apply role-based permissions on top of data in your collections.

Get Started

Usage This Month ⓘ

View docs for more info on tracking usage. You can also check the total usage for the project this app belongs to.

Requests

11.40K

Data Transfer

0.02 GB

Compute Runtime

0.70 Hours

Sync Runtime

0.00 Hours

# Authentication & Users

← 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

Feature Requests

Users

Custom User Data

Authentication Providers

Add New User

MongoDB Realm provides several authentication providers that you can integrate into a client application to allow users to log in to your Realm app. [Learn more about how to configure authentication for any of our authentication providers.](#)

Provider	Enabled	Edit
Allow users to log in anonymously	On	EDIT
Email/Password	On	EDIT
Facebook	Off	EDIT
Google	Off	EDIT
Apple	Off	EDIT
API Keys	Off	EDIT
Custom JWT Authentication	Off	EDIT
Custom Function Authentication	Off	EDIT

## Email/Password Settings

← 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

Feature Requests

Users

Custom User Data

Authentication Providers

Add New User

ONANDOFF > OMM > OMM > AUTH PROVIDERS > EDIT LOCAL-USERPASS

Provider Enabled

Allow your users to sign in with this authentication method.

ON

USER CONFIRMATION

User Confirmation Method

☐ Send a confirmation email

☐ Run a confirmation function

☒ Automatically confirm users

PASSWORD RESET

Password Reset Method

☒ Send a password reset email

☐ Run a password reset function

Password Reset URL

The Password Reset URL should point to a page that allows users to, at minimum, input a new password for their account. Realm will automatically append the user's token and tokenId to this URL in the email. The URL must include a scheme, such as http or https. [Learn how to support password resets here.](#)

https://www.google.de

Reset Password Email Subject

The subject line of the email sent to users when they request to reset their password. If this is not specified, Realm will use a default subject instead. The subject may have a maximum of 256 characters.

This is not implemented YET

No Changes

# Custom User Data

← 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

Users

Custom User Data

Authentication Providers

Add New User

Enable Custom User Data

ON

Store Custom User Data

Make it easy to access your user data within Functions and Rules. Enter in the cluster, database, and collection where your user data is stored. [Learn more about custom user data.](#)

Cluster Name

mongodb-atlas

Database Name

omm

Collection Name

users

User ID Field

Enter the field that will map the custom data to a user account ID

uid

No Changes

# Schemas & Relationships

## Comments

◀

MONGODB-ATLAS omm.comments

Rules Schema Relationships

EDIT SCHEMA VALIDATE

```
1 {
2   "title": "comment",
3   "properties": {
4     "_id": {
5       "bsonType": "objectId"
6     },
7     "meme_id": {
8       "bsonType": "objectId"
9     },
10    "text": {
11      "bsonType": "string"
12    },
13    "createdAt": {
14      "bsonType": "date"
15    },
16    "createdBy": {
17      "bsonType": "objectId"
18    }
19  }
20 }
```

◀

MONGODB-ATLAS omm.comments

Rules Schema Relationships Fi

```
1 {
2   "createdBy": {
3     "ref": "#/relationship/mongodb-atlas/omm/users",
4     "foreign_key": "_id",
5     "is_list": false
6   },
7   "meme_id": {
8     "ref": "#/relationship/mongodb-atlas/omm/memes",
9     "foreign_key": "_id",
10    "is_list": false
11  }
12 }
```



# Memes

← MONGODB-ATLAS omm.memes

RulesSchemaRelationships

EDIT SCHEMAVALIDATE

```
1 {
2   "title": "meme",
3   "properties": {
4     "_id": {
5       "bsonType": "objectId"
6     },
7     "commentCount": {
8       "bsonType": "int"
9     },
10    "comments": {
11      "bsonType": "array",
12      "items": {
13        "bsonType": "objectId"
14      }
15    },
16    "createdAt": {
17      "bsonType": "date"
18    },
19    "createdBy": {
20      "bsonType": "objectId"
21    },
22    "downVotes": {
23      "bsonType": "array",
24      "items": {
25        "bsonType": "objectId"
26      }
27    },
28    "forkedBy": {
29      "bsonType": "array",
30      "items": {
31        "bsonType": "objectId"
32      }
33    },
34    "forkedFrom": {
35      "bsonType": "objectId"
36    },
37    "isDraft": {
38      "bsonType": "bool"
39    },
40    "json": {
41      "bsonType": "string"
42    },
43    "points": {
44      "bsonType": "int"
45    },
46    "svg": {
47      "bsonType": "string"
48    },
49    "template": {
50      "bsonType": "objectId"
51    },
52    "title": {
53      "bsonType": "string"
54    },
55    "upVotes": {
56      "bsonType": "array",
57      "items": {
58        "bsonType": "objectId"
59      }
60    },
61    "url": {
62      "bsonType": "string"
63    },
```

← MONGODB-ATLAS omm.memes

RulesSchemaRelationshipsFilters

```
1 {
2   "comments": {
3     "ref": "#/relationship/mongodb-atlas/omm/comments",
4     "foreign_key": "_id",
5     "is_list": true
6   },
7   "createdBy": {
8     "ref": "#/relationship/mongodb-atlas/omm/users",
9     "foreign_key": "_id",
10    "is_list": false
11  },
12  "downVotes": {
13    "ref": "#/relationship/mongodb-atlas/omm/users",
14    "foreign_key": "_id",
15    "is_list": true
16  },
17  "forkedBy": {
18    "ref": "#/relationship/mongodb-atlas/omm/memes",
19    "foreign_key": "_id",
20    "is_list": true
21  },
22  "forkedFrom": {
23    "ref": "#/relationship/mongodb-atlas/omm/memes",
24    "foreign_key": "_id",
25    "is_list": false
26  },
27  "template": {
28    "ref": "#/relationship/mongodb-atlas/omm/templates",
29    "foreign_key": "_id",
30    "is_list": false
31  },
32  "upVotes": {
33    "ref": "#/relationship/mongodb-atlas/omm/users",
34    "foreign_key": "_id",
35    "is_list": true
36  }
37 }
```

# Templates

← MONGODB-ATLAS omm.templates

Rules Schema Relationships

EDIT SCHEMA VALIDATE

```
1 {
2   "title": "template",
3   "properties": {
4     "id": {
5       "bsonType": "objectId"
6     },
7     "createdAt": {
8       "bsonType": "date"
9     },
10    "createdBy": {
11      "bsonType": "objectId"
12    },
13    "height": {
14      "bsonType": "int"
15    },
16    "img": {
17      "bsonType": "string"
18    },
19    "type": {
20      "bsonType": "string"
21    },
22    "mediaType": {
23      "bsonType": "string"
24    },
25    "url": {
26      "bsonType": "string"
27    },
28    "width": {
29      "bsonType": "int"
30    },
31    "duration": {
32      "bsonType": "number"
33    },
34    "name": {
35      "bsonType": "string"
36    }
37  }
38 }
```

← MONGODB-ATLAS omm.templates

Rules Schema Relationships

```
1 {
2   "createdBy": {
3     "ref": "#/relationship/mongodb-atlas/omm/users",
4     "foreign_key": "_id",
5     "is_list": false
6   }
7 }
```

# Users

◀ MONGODB-ATLAS omm.users

Rules Schema Relationships

EDIT SCHEMA VALIDATE

```
1 {
2   "title": "user",
3   "properties": {
4     "_id": {
5       "bsonType": "objectId"
6     },
7     "createdAt": {
8       "bsonType": "date"
9     },
10    "email": {
11      "bsonType": "string"
12    },
13    "name": {
14      "bsonType": "string"
15    },
16    "uid": {
17      "bsonType": "string"
18    }
19  }
20 }
```

◀ MONGODB-ATLAS omm.users

Rules Schema Relationships

1 {}

# GraphQL

Get GraphQL URL here:

← Realm Apps

omm

DATA ACCESS

Rules

Schema

App Users

Authentication

BUILD

SDKs

Sync

GraphQL

ONANDOFF > OMM > OMM > GRAPHQL > EXPLORE

GraphQL

Explore Schema Custom Resolvers Settings

GraphQL Endpoint

Note: All requests to the GraphQL endpoint must be authenticated. [Learn how to retrieve and refresh access tokens.](#)

https://realm.mongodb.com/api/client/v2.0/app/omm-edgws/graphql

History × GraphQL

getCurrentMeme

1 # Welcome to Realm GraphQL!

Prettify Merge Copy History

Documentation Explorer ×

Search Schema...

# Custom Resolvers

← Realm Apps

omm

DATA ACCESS

Rules

Schema

App Users

Authentication

BUILD

SDKs

Sync

GraphQL

Functions

Triggers

3rd Party Services

Values

MANAGE

ONANDOFF > OMM > OMM > GRAPHQL > CUSTOM RESOLVERS

GraphQL

Explore Schema Custom Resolvers Settings

Add a Custom Resolver

Field Name	Parent Type	Linked Function	Actions
upVoteMeme	Mutation	upVoteMemeResolver	...
downVoteMeme	Mutation	downVoteMemeResolver	...
addView	Mutation	countViewResolver	...
fetchMeme	Query	fetchMemeResolver	...
fetchRandomMeme	Query	fetchRandomMemeResolver	...
addComment	Mutation	addCommentResolver	...
searchMemesByTitle	Query	searchMemesByTitleResolver	...

## Edit Custom Resolver: upVoteMeme

### GraphQL Field Name

This is the name of the field that will be injected into the type you select below.

upVoteMeme

### 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.

upVoteMemeResolver

### 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.

Custom Type

```
1 {
2   "type": "object",
3   "title": "VoteMemeInput",
4   "required": [
5     "meme_id",
6     "user_id"
7   ],
8   "properties": {
9     "meme_id": {
10      "bsonType": "objectId"
11    }
12  }
```

Ln 16 Col 2

- ✓ Must define valid JSON.
- ✓ Must define "type" as "object" or "array".
- ✓ Must define "title".
- ✓ Must define at least one field in "properties" with a "type".

### 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

### InputType:

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

omm



## upVoteMemeResolver

## DATA ACCESS

Rules

Schema

App Users

Authentication

## BUILD

SDKs

Sync

GraphQL

## Functions

Triggers

3rd Party Services

Values

## MANAGE

Linked Data Sources

Deploy

Hosting

Logs

## Function Editor

## Settings

```
1 exports = async function upVoteMeme({ meme_id, user_id }) {  
2   const cluster = context.services.get("mongodb-atlas");  
3   const memes = cluster.db("omm").collection("memes");  
4   await memes.updateOne(  
5     { _id: meme_id, upVotes: { $ne: user_id } },  
6     { $push: { upVotes: user_id },  
7       $pull: { downVotes: user_id },  
8     }  
9   );  
10  
11   const memeTmp = await memes.findOne(  
12     { _id: meme_id }  
13   )  
14  
15   const upVotes = memeTmp.upVotes != null ? memeTmp.upVotes.length : 0  
16   const downVotes = memeTmp.downVotes != null ? memeTmp.downVotes.length : 0  
17  
18   const pointsTmp = upVotes - downVotes  
19  
20   const meme = await memes.findOneAndUpdate(  
21     { _id: meme_id },  
22     { $set: {  
23       points: BSON.Int32(pointsTmp)  
24     } },  
25     { returnNewDocument: true }  
26   );  
27   return meme;  
28 }  
29  
30
```

← 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 EditorSettings

```
1 exports = async function upVoteMeme({ meme_id, user_id }) {
2   const cluster = context.services.get("mongodb-atlas");
3   const memes = cluster.db("omm").collection("memes");
4   await memes.updateOne(
5     { _id: meme_id, downVotes: { $ne: user_id } },
6     {
7       $push: { downVotes: user_id },
8       $pull: { upVotes: user_id },
9     }
10  );
11
12  const memeTmp = await memes.findOne(
13    { _id: meme_id }
14  )
15
16  const upVotes = memeTmp.upVotes != null ? memeTmp.upVotes.length : 0
17  const downVotes = memeTmp.downVotes != null ? memeTmp.downVotes.length : 0
18
19  const pointsTmp = upVotes - downVotes
20
21  const meme = await memes.findOneAndUpdate(
22    { _id: meme_id },
23    { $set: {
24      points: BSON.Int32(pointsTmp)
25    } },
26    { returnNewDocument: true }
27  );
28
29  return meme;
30 }
31
```



← 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 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

```
1 exports = async function countView(meme_id) {
2   const cluster = context.services.get("mongodb-atlas");
3   const memes = cluster.db("omm").collection("memes");
4   const meme = await memes.findOneAndUpdate(
5     { _id: meme_id },
6     {
7       $inc: { views: 1 },
8     },
9     { returnNewDocument: true }
10  );
11  return meme;
12 }
13
```



← 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

Feature Requests

ONANDOFF > OMM > OMM > GRAPHQL > CUSTOM RESOLVERS > EDIT FETCHMEME

Edit Custom Resolver: fetchMeme

GraphQL Field Name

This is the name of the field that will be injected into the type you select below.

fetchMeme

Parent Type

Select the type you would like your custom resolver to be accessed from.

Query

Function

Select the function that will be executed when a query including your custom resolver field is made.

fetchMemeResolver

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.

1 {

2   "type": "object",

3   "title": "FetchMemeInput",

4   "required": [

5     "meme\_id"

6   ],

7   "properties": {

8     "meme\_id": {

9       "bsonType": "objectId"

10    }

}

}

Ln 21 Col 2

✓ Must define valid JSON.

✓ Must define "type" as "object" or "array".

✓ Must define "title".

✓ Must define at least one field in "properties" with a "type".

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

## Input Type:

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

omm



## fetchMemeResolver

## 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

## Function Editor

## Settings

```
1 exports = async function fetchMeme({ meme_id, conditions, sorts, next }) {  
2  
3   const cluster = context.services.get("mongodb-atlas");  
4   const memeCollection = cluster.db("omm").collection("memes");  
5  
6   const sort = JSON.parse(sorts)  
7   const condition = JSON.parse(conditions)  
8  
9   if(condition.createdAt){  
10     for (var key of Object.keys(condition.createdAt)){  
11       condition.createdAt[key] = new Date(condition.createdAt[key])  
12     }  
13   }  
14   if(condition._id){  
15     for (key of Object.keys(condition._id)){  
16       condition._id[key] = new BSON.ObjectId(condition._id[key])  
17     }  
18   }  
19   if(condition.createdBy){  
20     for (key of Object.keys(condition.createdBy)){  
21       condition.createdBy[key] = new BSON.ObjectId(condition.createdBy[key])  
22     }  
23   }  
24  
25   if(condition.template){  
26     for (key of Object.keys(condition.template)){  
27       condition.template[key] = new BSON.ObjectId(condition.template[key])  
28     }  
29   }  
30  
31   const memes = await memeCollection.find(condition).sort(sort).toArray()  
32  
33   const index = memes.findIndex(x => x._id.toString() === meme_id.toString())  
34  
35   if(next){  
36     return index + 1 >= memes.length ? null : memes[index + 1]  
37   } else{  
38     return index - 1 < 0 ? null : memes[index - 1]  
39   }  
40  
41 }
```

← 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

Feature Requests

ONANDOFF > OMM > OMM > GRAPHQL > CUSTOM RESOLVERS > EDIT FETCHRANDOMMEME

Edit Custom Resolver: fetchRandomMeme

GraphQL Field Name

This is the name of the field that will be injected into the type you select below.

fetchRandomMeme

Parent Type

Select the type you would like your custom resolver to be accessed from.

Query

Function

Select the function that will be executed when a query including your custom resolver field is made.

fetchRandomMemeResolver

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.

1 {

2 "type": "object",

3 "title": "FetchRandomMemeInput",

4 "required": [

5 "meme\_id"

6 ],

7 "properties": {

8 "meme\_id": {

9 "bsonType": "objectId"

10 },

11 }

12 }

Ln 15 Col 2

✓ Must define valid JSON.

✓ Must define "type" as "object" or "array".

✓ Must define "title".

✓ Must define at least one field in "properties" with a "type".

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

## Input Type:

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

omm



## fetchRandomMemeResolver

### 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

Function Editor

Settings

```
1 exports = async function fetchRandomMeme({ meme_id, conditions }) {
2   const cluster = context.services.get("mongodb-atlas");
3   const memeCollection = cluster.db("omm").collection("memes");
4
5   const condition = JSON.parse(conditions)
6
7   if(condition.createdAt){
8     for (var key of Object.keys(condition.createdAt)){
9       condition.createdAt[key] = new Date(condition.createdAt[key])
10    }
11  }
12  if(condition._id){
13    for (key of Object.keys(condition._id)){
14      condition._id[key] = new BSON.ObjectId(condition._id[key])
15    }
16  }
17  if(condition.createdBy){
18    for (key of Object.keys(condition.createdBy)){
19      condition.createdBy[key] = new BSON.ObjectId(condition.createdBy[key])
20    }
21  }
22
23  if(condition.template){
24    for (key of Object.keys(condition.template)){
25      condition.template[key] = new BSON.ObjectId(condition.template[key])
26    }
27  }
28
29  const memes = await memeCollection.aggregate(
30    [
31      { $match: { _id: { $ne: meme_id } } },
32      { $match: condition },
33      { $sample: { size: 1 } }
34    ]
35  ).toArray()
36  return memes[0]
37 }
38
```

← 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

Feature Requests

ONANDOFF > OMM > OMM > GRAPHQL > CUSTOM RESOLVERS > EDIT ADDCOMMENT

Edit Custom Resolver: addComment

GraphQL Field Name

This is the name of the field that will be injected into the type you select below.

addComment

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.

addCommentResolver

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.

1 {

2   "type": "object",

3   "title": "AddCommentInput",

4   "properties": {

5     "meme\_id": {

6       "bsonType": "objectId"

7     },

8     "createdBy": {

9       "bsonType": "objectId"

10    },

}

}

Ln 18 Col 2

✓ Must define valid JSON.

✓ Must define "type" as "object" or "array".

✓ Must define "title".

✓ Must define at least one field in "properties" with a "type".

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

## Input Type:

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

omm



## addCommentResolver

### DATA ACCESS

Rules

Schema

App Users

Authentication

### BUILD

SDKs

Sync

GraphQL

### Functions

Triggers

3rd Party Services

#### Function Editor

#### Settings

```
1 exports = async function upVoteMeme(input) {
2   const cluster = context.services.get("mongodb-atlas");
3   const memes = cluster.db("omm").collection("memes");
4   const comments = cluster.db("omm").collection("comments");
5
6   const comment_id = (await comments.insertOne(input)).insertedId
7
8   const meme = await memes.findOneAndUpdate(
9     { _id: input.meme_id },
10    {
11      $push: { comments: comment_id },
12      $inc: { commentCount: 1 }
13    },
14    { returnNewDocument: true }
15  );
16  return meme;
17 }
18
```

← 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

Feature Requests

ONANDOFF > OMM > OMM > GRAPHQL > CUSTOM RESOLVERS > EDIT SEARCHMEMESBYTITLE

Edit Custom Resolver: searchMemesByTitle

GraphQL Field Name

This is the name of the field that will be injected into the type you select below.

searchMemesByTitle

Parent Type

Select the type you would like your custom resolver to be accessed from.

Query

Function

Select the function that will be executed when a query including your custom resolver field is made.

searchMemesByTitleResolver

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.

1 {

2 "type": "object",

3 "title": "SearchMemeInput",

4 "required": [

5 "searchString"

6 ],

7 "properties": {

8 "searchString": {

9 "bsonType": "string"

10 },

11 }

12 }

Ln 18 Col 2

✓ Must define valid JSON.

✓ Must define "type" as "object" or "array".

✓ Must define "title".

✓ Must define at least one field in "properties" with a "type".

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 (List)

[Meme]

From GraphQL Schema

No Changes

## Input Type:

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

omm



## searchMemesByTitleResolver

## 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

## Function Editor

## Settings

```
1 exports = async function fetchMeme({ searchString, conditions, sorts }) {  
2  
3   if(!searchString || searchString === '')  
4     return []  
5  
6   const cluster = context.services.get("mongodb-atlas");  
7   const memeCollection = cluster.db("omm").collection("memes");  
8  
9   const searchStringRegex = "(?i)" + searchString  
10  
11  const sort = JSON.parse(sorts)  
12  const condition = JSON.parse(conditions)  
13  
14  if(condition.createdAt){  
15    for (var key of Object.keys(condition.createdAt)){  
16      condition.createdAt[key] = new Date(condition.createdAt[key])  
17    }  
18  }  
19  if(condition._id){  
20    for (key of Object.keys(condition._id)){  
21      condition._id[key] = new BSON.ObjectId(condition._id[key])  
22    }  
23  }  
24  if(condition.createdBy){  
25    for (key of Object.keys(condition.createdBy)){  
26      condition.createdBy[key] = new BSON.ObjectId(condition.createdBy[key])  
27    }  
28  }  
29  
30  condition.title = { $regex: searchStringRegex }  
31  
32  const memes = await memeCollection.find(condition).sort(sort).toArray()  
33  
34  return memes  
35 }
```



← Realm Apps

02/28/2021 16:16:33 Deployment was successful!

ommm

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

Feature Requests

ONANDOFF > OMM > OMM > GRAPHQL > CUSTOM RESOLVERS > EDIT FETCHMEMEWITHPAGINATION

Edit Custom Resolver: fetchMemeWithPagination

GraphQL Field Name

fetchMemeWithPagination

Parent Type

Query

Function

fetchMemeWithPaginationResolver

Input Type (Recommended)

Custom Type

```

1 {
2   "type": "object",
3   "title": "FetchMemeWithPaginationInput",
4   "properties": {
5     "searchString": {
6       "bsonType": "string"
7     },
8     "conditions": {
9       "bsonType": "string"
10    },

```

- ✓ Must define valid JSON.
- ✓ Must define "type" as "object" or "array".
- ✓ Must define "title".
- ✓ Must define at least one field in "properties" with a "type".

Payload Type (Recommended)

Existing Type (List)

[Meme]

From GraphQL Schema

## Input Type:

```

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

```

omm



ONANDOFF > OMM > OMM > FUNCTIONS > FETCHMEMEWITHPAGINATIONRESOLVER

## fetchMemeWithPaginationResolver

No Changes



Function Editor

Settings

```
1 exports = async function fetchMemeWithPagination({ searchString, conditions, sorts, limit, skip }) {  
2  
3   const cluster = context.services.get("mongodb-atlas");  
4   const memeCollection = cluster.db("omm").collection("memes");  
5  
6   const sort = JSON.parse(sorts)  
7   const condition = JSON.parse(conditions)  
8  
9   if(condition.createdAt){  
10     for (var key of Object.keys(condition.createdAt)){  
11       condition.createdAt[key] = new Date(condition.createdAt[key])  
12     }  
13   }  
14   if(condition._id){  
15     for (key of Object.keys(condition._id)){  
16       condition._id[key] = new BSON.ObjectId(condition._id[key])  
17     }  
18   }  
19   if(condition.createdBy){  
20     for (key of Object.keys(condition.createdBy)){  
21       condition.createdBy[key] = new BSON.ObjectId(condition.createdBy[key])  
22     }  
23   }  
24  
25   if(searchString && searchString !== ""){  
26     const searchStringRegex = "(?!)"+searchString  
27     condition.title = { $regex: searchStringRegex }  
28   }  
29  
30   const memes = await memeCollection.find(condition).sort(sort).skip(skip).limit(limit).toArray()  
31  
32   return memes  
33 }  
34
```

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

# 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 need 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](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