Regenquest GraphQL API Documentation

Zeeshan Javed

Note: This is the first version of the database and the API, there will definitely be bugs/crashes. If something is missing or there is a bug. Message Zeeshan Javed on slack.

Please read this entire document before using the API. The API is built using **GraphQL.**

Below is the list of all the end points of the API and what they do. I have also provided examples of how to use each end point. All API responses will be in JSON format.

Below is the schema for each collection in the database.

Each Collection below also has the field “_ID” which is auto-generated by mongo DB. The API will not use the auto-generated _ID field for anything yet. We will use the custom id field in each collection.

Collection Name	**Collection Schema**
Users	{ userID: String, motive: String, biography: String, topics: [String], communities: [String], skills: [{skillName: String, skillLevel: String}], badges: [{badgeName: String, badgeDescription: String}], currentLevel: Number }
Tasks	{ taskID: String, userID: String, createdAt: String, name: String, description: String, requirements: [String], completionStatus: Boolean, history: [String] }
Quests	{ questID: String, name: String, type: String, duration: String, location: String, startDate: String, endDate: String, requirements: [String], members: [String] }
Posts	{ postID: String, userID: String, title: String, description: String, likes: Number, attachments: [String], createdAt: String, comments: [{ username: String, body: String }] }
Inventory	{ itemID: String, userID: String, taskLink: String, itemName: String, createdAt: String, description: String, image: String, history: [String] }
Events	{ eventID: String, name: String, theme: String, location: String, time: String, description: String, layer: String, hashtags: [String]}
Communities	{ communityID: String, name: String, description: String, members: [String], theme: String }

You can make 2 types of requests to this API. A **query** request and a **mutation** request.

A **query** request is used to **GET** data. Below is a list of all the query end points.

```
    getUsers: //returns a list of all users.
    getTasks: //returns a list of all tasks.
    getQuests: //returns a list of all quests.
    getPosts: //returns a list of all posts.
    getItems: //returns a list of all items.
    getEvents: //returns a list of all events.
    getCommunities: //returns a list of all communities.

    findUserbyID(userID: String): //returns user based on userID.
    findTaskbyID(taskID: String): //returns task based on taskID.
    findQuestbyID(questID: String): //returns quest based on questID.
    findPostbyID(postID: String)://returns post based on postID.
    findInventoryItembyID(itemID: String): //returns item based on itemID.
    findEventbyID(eventID: String): //returns event based on eventID.
    findCommunitybyID(communityID: String): //return community based on
communityID.
```

The first 7 end points listed above will return a list of the things requested. Since you are requesting all of the items, no input in required.

In the 2nd 7 end points listed above, you are trying to find a particular thing based on its ID, so an ID will need to be provided.

A **mutation** request is used to **create/update** data. Below is a list of all the mutation end points.

```
    createRegenquestUser(userInput: regenquestUserInput):
    createRegequestTask(userInput: regenquestTaskInput):
    createRegenquestQuest(userInput: regenquestQuestInput):
    createRegenquestPost(userInput: regenquestPostInput):
    createRegenquestInventory(userInput: regenquestInventoryInput):
    createRegenquestEvent(userInput: regenquestEventInput):
    createRegenquestCommunity(userInput: regenquestCommunityInput):

    updateRegenquestUser(userInput: regenquestUserInput):
    updateRegenquestTask(userInput: regenquestTaskInput):
    updateRegenquestQuest(userInput: regenquestQuestInput):
    updateRegenquestPost(userInput: regenquestPostInput):
    updateRegenquestInventory(userInput: regenquestInventoryInput):
    updateRegenquestEvent(userInput: regenquestEventInput):
    updateRegenquestCommunity(userInput: regenquestCommunityInput):
```

The first 7 end points listed above are used to create a new document. User input will need to be provided. I will show an example of how this is done later in this document. The schema at the top of this documents shows all of the available fields in a document. When creating a document if you do not provide input for a field, its default value will be set to null. This is can later be changed by using the update end points.

The second 7 end points listed above are used to update data in an existing document. User input will need to be provided for updating data. The following fields can not be changed after the document is created. [userID, taskID, questID, PostID, itemID, eventID, communityID]. Remember these id fields are different from the default ones that get auto-generated by mongo DB. The API does not user the auto-generated _ID fields.

How to use the API

Query requests

Example 1

When making query requests to the API, the request **MUST** be sent as a **GET** request.

API gateway link = https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest

Key = provided separately

When making **query requests** all input must be passed as parameters in the link.

For example:

If you want to get a list of all Users. You need to make the following request to the getUsers end point.

GET https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest?key=<key>&query=<query>

<Key> = for now assume key is "mykey123"

<query> = query{getUsers{**userID currentLevel**}}

Not substitute the variables in the link

GET https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest?key=mykey123&query= query{getUsers{userID currentLevel}}

The above request will return a list of all users with the following data, {userID, currentLevel}

If you want to receive additional about the users. You can modify the query as follow:

<query> = query{getUsers{**userID currentLevel biography topics communities**}}

You can add any of the schema fields into the query and you will receive corresponding data.

You must request at least one field.

You can request data about the Tasks, Events, Communities, Quests, or Posts by simply changing "getUsers" from the query to "getTasks", "getPosts", … etc.

If you wanted to get the names and IDs of all tasks you would make the following request.

GET https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest?key=mykey123&query= query{getTasks{taskID name}}

This will return a list of tasks with there name and ID. You can add additional fields to get more data. Remember Query request must be sent as **GET** requests. Also make sure to add the following header to all your requests content-type: application/JSON

Example 2

What if you wanted to find a user by there userID. Remember userID is a custom id field, different from _ID.

Since you are looking for a particular user, you need to pass there userID as a parameter.

Again, remember query requests are sent as **GET** requests.

userID = k443h3 Note: this is the userID of a test user I added to the DB, the user may be deleted by now.

GET https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest?query=query{findUserbyID(userID:"k443h3"){currentLevel motive userID}}&key=mykey123

The above request will return the following:

```
{
    "data": {
        "findUserbyID": {
            "currentLevel": 2555,
            "motive": "my  new motive",
            "userID": "k443h3"
        }
    }
}
```

You can use the above request to get any particular user, post, event, .. etc. Just remember to change “findUserbyID” to the appropriate value. All of the end points are listed at the top of this document.

Mutation Requests

Mutation requests work differently from query requests. Mutation requests are used to create/update/delete data. There are no delete end points in the API yet. But create and update exist.

Mutation requests MUST be sent as **POST** requests.

When you are creating a user you need to provide input fields, these fields must be provided in JSON format in the **body** of the request. Mutation requests are always sent as JSON in the body.

For example, imagine you want to create a new user:

You would send the following POST request.

POST https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest?key=mykey123

Attach the below **JSON** object to the **body** of the request.

```
{
    "query": "mutation Mutation($userInput: regenquestUserInput) {createRegenquestUser
(userInput: $userInput) {code response}}",
    "variables": {
  "userInput": {
    "badges": [
      {
        "badgeName": "badge 1",
        "badgeDescription": "badge 1 desc"
      },
      {
          "badgeName": "badge 2",
          "badgeDescription": "badge 2 desc"
      }
    ],
    "biography": "my bio",
    "communities": ["jh4jk3443jkh", "kj34j34jk"],
    "currentLevel": 12,
    "motive": "my motive",
    "skills": [
      {
        "skillName": "skill 1",
        "skillLevel": "advanced"
      }
    ],
    "topics": ["apple", "orange"],
    "userID": "k443h3"
  }
}
}
```

In the “userInput” object, you can delete any field you want, any field that is not provided will be changed to the default value of null. You can follow the above example to create user, post, event, .. etc.

For example, if you wanted to create a post you would send a POST request to the following address.

POST https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest?key=mykey123

Attach the below **JSON** object to the **body** of the request.

```
{
    "query": "mutation Mutation($userInput: regenquestPostInput) {createRegenquestPost
(userInput: $userInput) {code response}}",
    "variables": {
  "userInput": {
      "postID": "h34hj43hj34",
      "userID": "jhhjhxj5hj3",
      "title": "my first post",
      "description": "my first post desc",
      "attachments": ["attachment 1", "attachment 2"]
  }
}
}
```

Both of the above example will return a JSON object that has a 'code' and 'response' field that tells you if the action was successful.

```
{
    "data": {
        "createRegenquestPost": {
            "code": 0,
            "response": "successful"
        }
    }
}
```

Example 2 What if you wanted to update a document.

Updating also requires a POST request, with input as JSON format in the body.

For example, assume you wanted to change the current level of a user. You would make following request.

POST https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest?key=mykey123

Attach the below **JSON** object to the **body** of the request.

```
{
    "query": "mutation Mutation($userInput: regenquestUserInput) {updateRegenquestUser
(userInput: $userInput) {code response}}",
    "variables": {
  "userInput": {
      "userID": "k443h3",
      "currentLevel": 12
        }
    }
}
```

Any variable that you do not add in “userInput” will remain the same.

userID must always be provided when updating a document. The API uses the userID to find and update the document. If you are updating a post change the userID to postID … etc.

if you wanted to update multiple fields simply list them in the “userInput”

POST https://my-gateway-6qkoui4z.uc.gateway.dev/regenquest?key=mykey123

Attach the below **JSON** object to the **body** of the request.

```
{
    "query": "mutation Mutation($userInput: regenquestUserInput) {updateRegenquestUser
(userInput: $userInput) {response}}",
    "variables": {
  "userInput": {
    "biography": "my new  new bio",
    "currentLevel": 2555,
    "motive": "my  new motive",
    "userID": "k443h3"
  }
}
}
```

You can use the same method shown above to create/update any document.

Note: when creating/updating the documents the “createdAt” field is filled in my the API you do not need to provide this variable. When creating a Post, the “likes” field is set to 0 by the API, you do not need to provide this field when creating a post.

Please message Zeeshan Javed on slack if you need help with this API.