The message service maintains a relationship between topcoder entities, such as projects, challenges, and submissions with Discourse topics.

This relationship is maintained by the topics table (app/models/topics.js).

Currently, the message service does several things when users try to get a topic:

1. First it checks if a topic associated with the requested entity (reference and referenceId) exists, if it does:
    a. It tries to get a topic from Discourse using the user's handle. If the topic can be retrieved it means that the user has seen it previously, and has access to the topic, so it is returned.
    b. If the topic cannot be retrieved by the user from Discourse, it is possible that either the user has not been provisioned in Discourse yet, or it doesn't have access to the topic in Discourse, so it:
        i. Checks if the user has access to the entity on the Topcoder side by making a call to the endpoint registered in the referenceLookup Postgres table, if the user doesn't have access, returns 403, otherwise continue;
        ii. Tries to fetch the user from Discourse, if the user doesn't exist, it creates a user in Discourse;
        iii. Tries to fetch the topic from Discourse, if the topic doesn't exist, it creates one.
        iv. Gives access to the user to see the Discourse topic;
        v. Returns the topic;
2. If the reference doesn't exist:
    a. Checks if the user has access to the entity on the Topcoder side by making a call to the endpoint registered in the referenceLookup Postgres table, if the user doesn't have access, returns 403, otherwise continue;
    b. Tries to fetch the user from Discourse, if the user doesn't exists it creates the user in Discourse;
    c. Creates the topic in Discourse giving user access to it;
    d. Registers the topic in the topic Postgres table;
    e. Returns the topic;

Since we are building a new endpoint to create topics, we no longer want the GET topics api to create topics automatically. The first task is to modify the get topic api to remove all instances where it creates a topic. If a topic doesn't exist 404 should be returned with the body:

```
{
  "message": "Topic doesn't exists"
}
```

Next, you will notice that based on this data model, it is possible to have multiple topics per topcoder entity, e.g. submission 123 could have multiple topics in the table above in which reference = submission, and referenceId = 123.

However, the current implementation of the GET /v4/topics assumes a single topic per entity, and always returns the first topic that it finds.

The second task is to expand this implementation, and allow the support of multiple topics per specific entity.

To differentiate between topics, we are going to start using the "tag" field of the topic table shown above.

A topic will only have one tag. There may be multiple topics with the same tag.

We need to make sure we allow filtering by the tag field. We need to add support for the field tag to be used as part of the filter in the get topics endpoint e.g.:

GET /v4/topics?reference%3Dsubmission%26referenceId%3D123%26**tag%3DPRIMARY**

The above API call should return all topics that match the reference, referenceId, and tag. If there are multiple topics, they should be returned in order of the last activity date descending (more recent activity first). All topics should be returned as a json array, and the value of the tag field should be injected in each topic object.

Finally we will need an endpoint that can will allow us to create new topics:

POST /v4/topics

Headers:
Authorization: Bearer {TOKEN}
Content-Type: application/json

Body:

```
{
  "reference": "submission" | "challenge" | "project" | "etc.",
  "referenceId": "12345",
  "tag": "ANYTAG",
  "title": "This is the tile of the message",
  "body": "This is the body of the message"
}
```

The request above should perform the following steps:

- Verify if the user has access to the entity (userHasAccessToEntity function), if the user doesn't have access return 403;
- Try to create a private post in Discourse (createPrivatePost in discourse.js)
- If it fails, check if the user exists in Discourse, if the user doesn't exist, provision it, and try to create the private post again.
- Set system, and the current user as the users in the post;
- Return the newly created topic