

# GraphQL Mutations

In this lesson, we will be introducing the GraphQL mutation and actually make one ourselves.

## WE'LL COVER THE FOLLOWING ^

- Mutation vs. Query
- Starring a GitHub Repository

## Mutation vs. Query #

GraphQL mutations complement GraphQL queries because they are used for writing data that queries can read. The mutation shares the same principles as the query: it has fields and objects, arguments and variables, fragments and operation names, as well as directives and nested objects for the returned result. When the mutation is valid, we should receive the updated data for the specified fields and objects in our query. Before we start making our first mutation, we need to be aware that we are using live GitHub data, so if you follow a person on GitHub using your experimental mutation, you will follow this person for real.

## Starring a GitHub Repository #

In this lesson, we will **star** a repository on GitHub using a mutation from GitHub's API. This would be the same repository we requested using a query in the previous lesson. The **addStar** mutation can be found in the "Docs" sidebar. The repository we will be working with is a project for teaching developers about the fundamentals of React so, starring it should prove useful.

You can visit [the repository](#) just to make sure that you haven't given it a star already. We want an unstarred repository so that we can star it using a mutation. Before you can star a repository, you need to know its identifier which can be retrieved by a query:

Environment Variables 

Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
query {  
  organization(login: "the-road-to-learn-react") {  
    name  
    url  
    repository(name: "the-road-to-learn-react") {  
      id  
      name  
    }  
  }  
}
```



In the results section in GraphiQL for the above query, we should see the identifier for the repository: `"MDEwO1JlcG9zaXRvcnk2MzM1MjkwNw=="`. Before using the identifier as a variable, we can structure your mutation in GraphiQL the following way:


Environment Variables 

Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
mutation AddStar($repositoryId: ID!) {  
  addStar(input: { starrableId: $repositoryId }) {  
    starrable {  
      id  
      viewerHasStarred  
    }  
  }  
}
```




The mutation's name is given by GitHub's API: `addStar`. We are required to pass it the `starrableId` as input to identify the repository; otherwise, the GitHub server won't know which repository to star with the mutation. In addition to this, the mutation is a named mutation: `AddStar`; it's up to us to give it a name. Lastly, we can define the return values of the mutation by using objects and fields again; identical to a query. Finally, the variables tab provides the variable for the mutation you retrieved with the last query:


Environment Variables 

Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
{  
  "repositoryId": "MDEwO1JlcG9zaXRvcnk2MzM1MjkwNw=="  
}
```



Once we execute the mutation, the result should look like the following. Since we specified the return values of our mutation using the `id` and `viewerHasStarred` fields, we should see them in the result.

Environment Variables 

Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
{  
  "data": {  
    "addStar": {  
      "starrable": {  
        "id": "MDEwO1JlcG9zaXRvcnk2MzM1MjkwNw==",  
        "viewerHasStarred": true  
      }  
    }  
  }  
}
```



We have successfully starred the repository. It's visible in the result, but you can also verify it in the repository on GitHub. Congratulations, you just made your first mutation!

Let's now move on to pagination.