



# Authorization

## Contents

- Lesson Overview (#\_lesson\_overview)
- The @auth Directive (#\_the\_auth\_directive)
- Setup (#\_setup)
- JSON Web Token (JWT) (#\_json\_web\_token\_jwt)
- Adding Authorization Rules (#\_adding\_authorization\_rules)
- Exercise: Adding A Customer & Order (#\_exercise\_adding\_a\_customer\_order)
- Check Your Understanding (#\_check\_your\_understanding)
- Summary (#\_summary)

## Lesson Overview

In this lesson we introduce authorization, protecting objects in our API. We cover how to make data available only to authenticated users and how to restrict data access so that customers can only see or modify their own data.

## The @auth Directive

The Neo4j GraphQL Library provides an `@auth` GraphQL schema directive that enables us to attach authorization rules to our GraphQL type definitions. The `@auth` directive uses JSON Web Tokens (JWTs) for authentication. Authenticated requests to the GraphQL API will include an `authorization` header with a Bearer token attached. For example:

HTTP

```
POST / HTTP/1.1
authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJyb2x1cyI6WyJ1c2VyX2FkbWluIiwicG9zdF9hZG1pbiIsImdyb3VwX2FkbWluI19.IY0LWqgHcjEt0s0w60mqKazhuRFKroSXFQkpCtWpgQI
content-type: application/json
```

Refer to the [Auth section of the Neo4j GraphQL Library documentation](#) for full information.

[Need help? Ask in the Neo4j Community.](#)

## Setup

To enable authorization with the Neo4j GraphQL Library we need to specify the JWT signing secret used to decode and validate tokens. For this lesson we will use the following JWT secret to validate our tokens:

[Copy to Clipboard](#)

```
dFt8QaYykR6PauvxcyKVXKauxvQuWQTc
```

When instantiating `Neo4jGraphQL` pass a config object that includes the JWT secret. For example:

JavaScript

```
const neoSchema = new Neo4jGraphQL({
  typeDefs,
  config: {
    jwt: {
      secret: "dFt8QaYykR6PauvxcyKVXKauxvQuWQTc"
    }
  }
});
```

This step has been completed in the initial Codesandbox we'll use for this lesson. Open [the Codesandbox using this link](#) [↗](#), again forking and editing the `.env` file to add the connection credentials for your Neo4j Sandbox instance. You'll notice we've included an additional environment variable to keep track of our JWT secret.

## JSON Web Token (JWT)

JWTs are a standard for representing and cryptographically verifying claims securely and are commonly used for authentication and authorization. Implementing a sign-in flow and generating JWTs is beyond the scope of this course so we will use a few static JWTs for testing our authorization rules. For an example of a sign-up/sign-in flow using GraphQL mutations and the Neo4j GraphQL Library see the [neo-push example application](#) [↗](#) in the `neo4j/graphql` repository.

### Example JWTs

We will use the following JWTs to test the authorization rules we'll be adding to our GraphQL API.

These tokens were generated using the JWT secret signing key above and can be validated using the same secret.

### Token For Customer EmilEifrem7474

This token is the token we will use to make authenticated requests on behalf of the customer "EmilEifrem7474":

Copy to Clipboard

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJFbWlsRWlmcmVtNzQ3NCIsInJvbGVzIjpbImN1c3RvbWVyI10sIm1hdCI6MTUxNjIzOTAyMn0.YwftAMDTw6GqmY0FLGHC_f6UiUhfrJAGkZGfrGmiQ2U
```

The token's payload includes the following claims:

JSON

```
{
  "sub": "EmilEifrem7474",
  "roles": ["customer"],
  "iat": 1516239022
}
```

### Admin user token

This token is used to make authenticated requests to the GraphQL API as an "admin" user:

Copy to Clipboard

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJCb2JMb2JsYXc3Njg3Iiwicm9sZXMiOiJWRtaW4iXSwiYW50IjoxNTE2MjM5MDIyfQ.f2GKIu31gz39fMJwj5_byFCMDPDy3ncdW0IhhqcwBxk
```

It includes the following claims:

JSON

```
{
  "sub": "BobLoblaw7687",
  "roles": ["admin"],
  "iat": 1516239022
}
```

Need help? [Ask in the Neo4j Community](#) ↗

We can use the online tool at [jwt.io](https://jwt.io) to encode/decode and validate tokens. Try pasting one of the above tokens into this tool to view the token's payload.

**Encoded**
PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJCb2Jmb2JsYXc3Njg3Iiwicm9sZXMiOiJsiYWRTaW4iXSwiaWF0IjoxNTE2MjM5MDIyfQ.f2GKIu31gz39fMJwj5_byFCMDPDy3ncdW0IhhqcwBxk
```

**Decoded**
EDIT THE PAYLOAD AND SECRET

**HEADER: ALGORITHM & TOKEN TYPE**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

**PAYLOAD: DATA**

```
{
  "sub": "BobLoblaw7687",
  "roles": [
    "admin"
  ],
  "iat": 1516239822
}
```

**VERIFY SIGNATURE**

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  dFt8QaYkR6PauvxcyKV
) ☐ secret base64 encoded
```

**Signature Verified**

**SHARE JWT**

## Adding Authorization Rules

Now that we have our sample tokens we're ready to start adding authorization rules using the `@auth` schema directive in our GraphQL type definitions.

### isAuthenticated

The `isAuthenticated` rule is the simplest authorization rule we can add. It means that any GraphQL operation that accesses a field or object protected by the `isAuthenticated` rule must have a valid JWT in the request header.

Let's make use of the `isAuthenticated` authorization rule in our bookstore GraphQL API to protect the `Subject` type. Let's say we want to make returning a book's subjects a "premium" feature to encourage users to sign-up for our application. To do this we'll make the following addition to our GraphQL type definitions, extending the `Subject` type:

GraphQL

Copy to Clipboard

```
# schema.graphql

extend type Subject @auth(rules: [{isAuthenticated: true}])
```

Need help? [Ask in the Neo4j Community](#)

Now any request that accesses the **Subject** type must include a valid signed JWT or an error will be returned.

Here we query as usual without including a JWT in the request header:

The screenshot shows the GraphQL Playground interface. On the left, the query is:
 

```
1 {
2   books {
3     title
4     subjects {
5       name
6     }
7   }
8 }
9
```

 The right pane shows the JSON response:
 

```
{
  "errors": [
    {
      "message": "Unauthenticated",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "books"
      ],
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "exception": {
          "stacktrace": [
            "Neo4jGraphQLAuthenticationError: Unauthenticated",
            "    at Object.<anonymous> (/sandbox/node_modules/@neo4j/graphql/dist/utils/excute.js:74:23)",
            "    at Generator.throw (<anonymous>)",
          ]
        }
      }
    }
  ]
}
```

 The bottom left pane shows 'QUERY VARIABLES' and 'HTTP HEADERS' (empty). The bottom right pane shows 'TRACING' and 'QUERY PLAN'.

In GraphQL Playground to add a request header we add the following to the "HTTP Headers" box in the lower left window. Now that our request includes a valid token, the data is returned as expected:

The screenshot shows the GraphQL Playground interface with the same query as before. The right pane shows the JSON response:
 

```
{
  "data": {
    "books": [
      {
        "title": "Graph Algorithms",
        "subjects": [
          {
            "name": "Non-fiction"
          },
          {
            "name": "Neo4j"
          },
          {
            "name": "Graph theory"
          }
        ]
      },
      {
        "title": "Inspired",
        "subjects": [
          {
            "name": "Non-fiction"
          }
        ]
      }
    ]
  }
}
```

 The bottom left pane now shows 'HTTP HEADERS (1)' with the following content:
 

```
1 {
2   "authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9"
3 }
```

 The bottom right pane shows 'TRACING' and 'QUERY PLAN'.

## Roles

**Roles** are the next type of authorization rule that we will explore. A JWT payload can include an array of "roles" that describe the permissions associated with the token. For example, our example token for the "admin" user includes the following payload:

Need help? [Ask in the Neo4j Community](#)

## JSON

```
{
  "sub": "BobLoblaw7687",
  "roles": ["admin"],
  "iat": 1516239022
}
```

which means that this user has the "admin" role. Let's add a rule to our GraphQL type definitions that in order to create, update, or delete books, the user must have the "admin" role.

## GraphQL

Copy to Clipboard

```
# schema.graphql

extend type Book @auth(rules: [{operations: [CREATE, UPDATE, DELETE], roles: ["admin"]}])
```

Note that we've included the **operations** array to specify this rule only applies to **CREATE**, **UPDATE**, and **DELETE** operations - all users will still be able to read book objects, but if any request tries to create or update a book the operation will fail unless a valid "admin" role is included in the token.

Try to execute the following mutation using our example tokens. What error do you get when not using the admin token? What is the result when using the admin token?

## GraphQL

Copy to Clipboard

```
mutation {
  createBooks(
    input: {
      title: "Graph Databases"
      isbn: "1491930896"
      subjects: { connect: { where: { name: "Neo4j" } } }
    }
  ) {
    books {
      title
      subjects {
        name
      }
    }
  }
}
```

Need help? [Ask in the Neo4j Community](#) ↗



TRACING QUERY PLAN

Of course we will also allow admins to have access to orders, so let's update the rule to also grant access to any requests with the "admin" role:

GraphQL

Copy to Clipboard

```
# schema.graphql

extend type Order @auth(rules: [{allow: {customer: {username: "$jwt.sub"}}}, {roles: ["admin"]}])
```

## Where

In the previous example the client was required to filter for orders that the customer had placed. We don't always want to expect the client to include this filtering logic in the GraphQL query. In some cases we simply want to return whatever data the currently authenticated user has access to. For these cases we can use a [Where](#) authorization rule to apply a filter to the generated database queries - ensuring only the data the user has access to is returned.

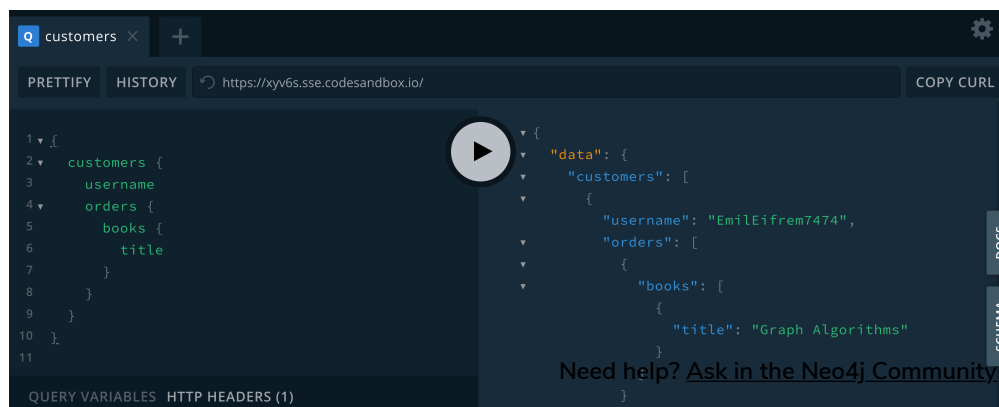
We want a user to only be able to view their own customer information. Here we add a rule to the **Customer** type that will apply a filter any time the customer type is accessed that filters for the currently authenticated customer by adding a predicate that matches the **username** property to the **sub** claim in the JWT.

GraphQL

Copy to Clipboard

```
# schema.graphql

extend type Customer @auth(rules: [{where: {username: "$jwt.sub"}}])
```





```

1 {
2   "authorization": "Bearer eyJhbGciOiJIUzI1NiIsI
3 }

```

TRACING QUERY PLAN

Note that our query doesn't specify which customer to return - we're requesting all customers - but we only get back the customer that we have access to.

## Bind

The final type of authorization rule that we will explore is the [Bind](#) rule. Bind allows us to specify connections that must exist in the graph when creating or updating data based on claims in the JWT.

We want to add a rule that when creating a review, the review node is connected to the currently authenticated customer - we don't want customers to be writing reviews on behalf of other users! This rule means the username of the author of a review must match the **sub** claim in the JWT when creating or updating reviews:

GraphQL

Copy to Clipboard

```

# schema.graphql

extend type Review @auth(rules: [{operations: [CREATE,UPDATE], bind: {author: {username:
"$jwt.sub"}} }])

```

If a customer tries to create a review and connect it to a customer other than themselves the mutation will return an error. Try running this mutation using our example JWT. Does it work? Can you tell why?

GraphQL

Copy to Clipboard

```

mutation {
  createReviews(
    input: {
      rating: 1
      text: "Borrring"
      book: { connect: { where: { title: "Ross Poldark" } } }
      author: { connect: { where: { username: "BookLover123" } } }
    }
  ) {
    reviews {
      text
      rating
      book {
        title

```

Need help? [Ask in the Neo4j Community](#)

```

    }
  }
}

```

## Auth In Cypher Directive Fields

There are two ways to make use of authorization features when using the `@cypher` schema directive:

1. Apply the authorization rules `isAuthenticated` and `roles` using the `@auth` directive.
2. Reference the JWT payload values in the Cypher query attached to a `@cypher` schema directive.

Let's make use of both of those aspects by adding a Query field that returns personalized recommendations for a customer. In our Cypher query we'll have access to a `$auth.jwt` parameter that represents the payload of the JWT. We'll use that value to look up the currently authenticated customer by username, then traverse the graph to find relevant recommendations based on their purchase history. We'll also include the `isAuthenticated` rule since we only want authenticated customers to use this Query field.

GraphQL

Copy to Clipboard

```

# schema.graphql

extend type Query {
  booksForCurrentUser: [Book] @auth(rules: [{ isAuthenticated: true }]) @cypher(statement:
"""
MATCH (c:Customer {username: $auth.jwt.sub})-[:PLACED]->(:Order)-[:CONTAINS]->(b:Book)
MATCH (b)-[:ABOUT]->(s:Subject)-[:ABOUT]->(rec:Book)
WITH rec, COUNT(*) AS score ORDER BY score DESC
RETURN rec
""")
}

```

Try running this GraphQL query. What results do you get?

GraphQL

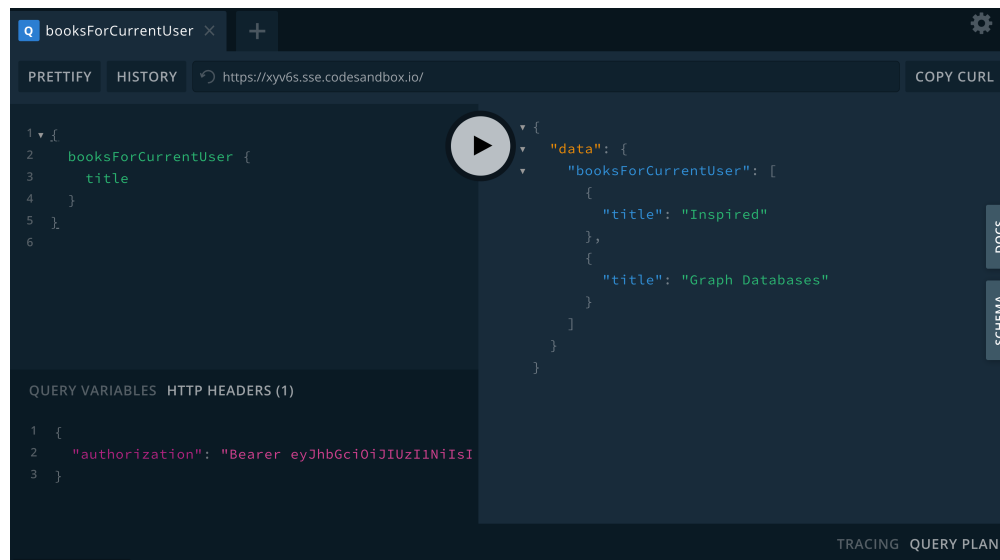
Copy to Clipboard

```

{
  booksForCurrentUser {
    title
  }
}

```

Need help? [Ask in the Neo4j Community](#) ↗



## Exercise: Adding A Customer & Order

1. Update the `schema.graphql` file adding in the authorization rules we covered above. If you get stuck [this Codesandbox](#) includes all the code from this lesson.
2. Using the admin token, create a new user.
3. Next, create a JWT token for this user using [jwt.io](#). Use this token to create an order for this user. Be sure to include some books in the order!
4. Next, add a review for the book purchased by this user.
5. Finally, write a query to view the customer's details, including their order history and their reviews.

## Check Your Understanding

### Question 1

Decode the following JWT using [jwt.io](#) or another method and inspect the payload of the token. What is the value of the `sub` claim on this token?

Copy to Clipboard

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJKZW5ueUNhdDQwNiIsIm1hdCI6MTUxNjIzOTAyMn0.rS9h2wbNJDp5mBtj_Of2-I9KnkaMa8xi63n0cFN40bs
```

Choose the correct answer.

Need help? [Ask in the Neo4j Community](#)

- ☐ EmilEifrem7474
- ☐ BookLover123
- ☐ JennyCat406
- ☐ BobLoblaw123

## Question 2

Which of the following GraphQL SDL snippets show an authorization rule that states only a user with the role `admin` can create `User` nodes.

Select the correct answer.

- ☐ `extend type User @auth(rules: [{operations: [CREATE], roles: ["admin"]}])`
- ☐ `extend type User @auth(rules: [{operations: [CREATE], allow: {role: admin}}])`
- ☐ `CREATE (u:User) WHERE u.role = "admin"`

## Question 3

When using the `@cypher` schema directive to define custom logic in the GraphQL schema there is no way to make use of the `@auth` directive to apply authorization rules for fields that use the `@cypher` directive.

Is this statement true or false?

- ☐ True
- ☐ False

## Summary

In this lesson, we explored how to add authorization rules to our GraphQL API using the `@auth` schema directive. In the next lesson we explore using the Neo4j GraphQL OGM.

✓ [Check Answers](#)

Need help? [Ask in the Neo4j Community](#)

Prev

◀ Adding Custom Logic

Next

The Neo4j GraphQL OGM ▶

Contents

Lesson Overview (#\_lesson\_overview)

The @auth Directive (#\_the\_auth\_directive)

Setup (#\_setup)

JSON Web Token (JWT) (#\_json\_web\_token\_jwt)

Adding Authorization Rules (#\_adding\_authorization\_rules)

Exercise: Adding A Customer & Order (#\_exercise\_adding\_a\_customer\_order)

Check Your Understanding (#\_check\_your\_understanding)

Summary (#\_summary)

© 2021 Neo4j, Inc.

Terms (https://neo4j.com/terms/).

| Privacy (https://neo4j.com/privacy-policy/).

| Sitemap (https://neo4j.com/sitemap/).

Neo4j<sup>®</sup>, Neo Technology<sup>®</sup>, Cypher<sup>®</sup>, Neo4j<sup>®</sup> Bloom<sup>™</sup> and Neo4j<sup>®</sup> Aura<sup>™</sup> are registered trademarks of Neo4j, Inc. All other marks are owned by their respective companies.

Contact Us → (https://neo4j.com/contact-us/?ref=footer).


US: 1-855-636-4532

Sweden +46 171 480 113


UK: +44 20 3868 3223

France: +33 (0) 8 05 08 03 44


Learn



Sandbox (https://neo4j.com/sandbox/?ref=developer-footer).




Neo4j Community Site (https://community.neo4j.com?ref=developer-footer).




Neo4j Developer Blog


Social




Twitter (https://twitter.com/neo4j).




Meetups (https://www.meetup.com/Neo4j-Online-Meetup/).



Github (https://github.com/neo4j/neo4j).



Stack Overflow (https://stackoverflow.com/questions/tagged/neo4j).




Need help? Ask in the Neo4j Community. ➤

13 of 14

7/2/21, 22:43

(<https://medium.com/neo4j>).

 [Neo4j Videos \(https://www.youtube.com/neo4j\)](https://www.youtube.com/neo4j)

 [GraphAcademy \(https://neo4j.com/graphacademy/?ref=developer-footer\)](https://neo4j.com/graphacademy/?ref=developer-footer)

 [Neo4j Labs \(https://neo4j.com/labs/?ref=developer-footer\)](https://neo4j.com/labs/?ref=developer-footer)

Want to Speak? [Get \\$ back. \(https://neo4j.com/speaker-program/\)](https://neo4j.com/speaker-program/)

Need help? [Ask in the Neo4j Community](#) ↗