In this example, we will use **jQuery to create a client application**.

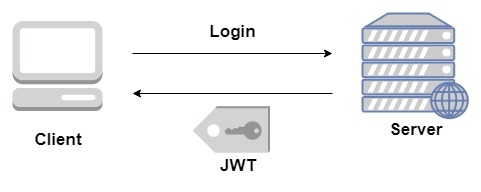
To authenticate requests, we will use **express-jwt** module on the server-side.

The express-jwt module is a middleware that lets you authenticate HTTP requests using JWT tokens. JSON Web Token (JWT) is a long string that identifies the logged in user.

Once the user logs in successfully, **the server generates a JWT token**.

This token distinctly identifies a log. In other words, the token is a representation of user's identity. So next time, when the client comes to the server, it has to present this token to get the required resources.

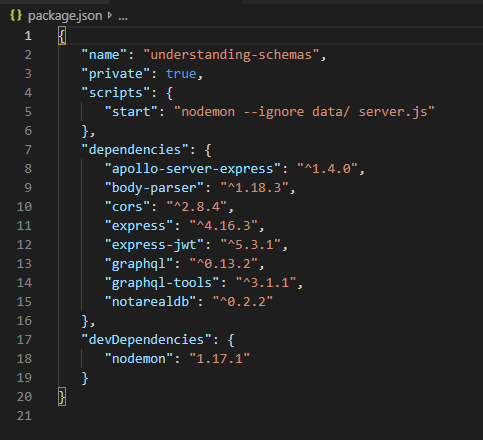
The client can be either a mobile application or a web application.

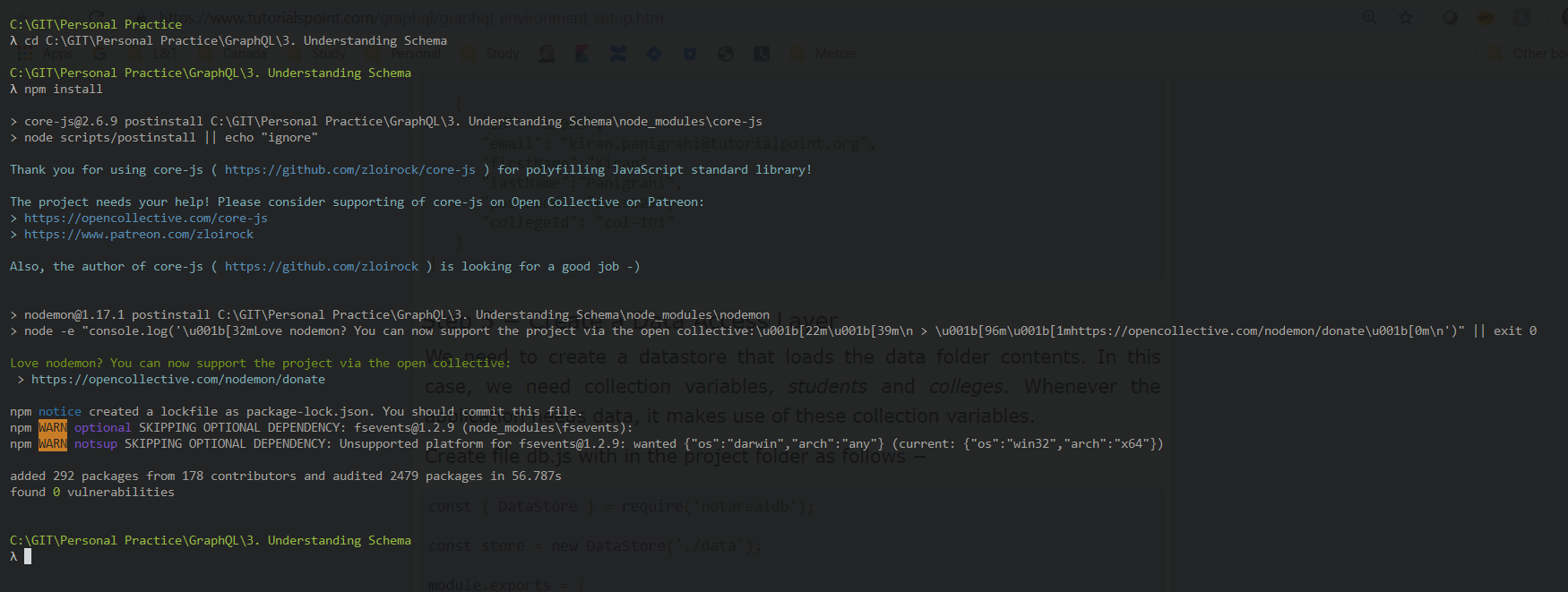


# Setting up the Server

## Step 1 − Download and Install Required Dependencies for the Project

Create new package.json file with below code and install all dependencies.

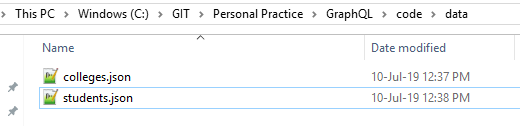


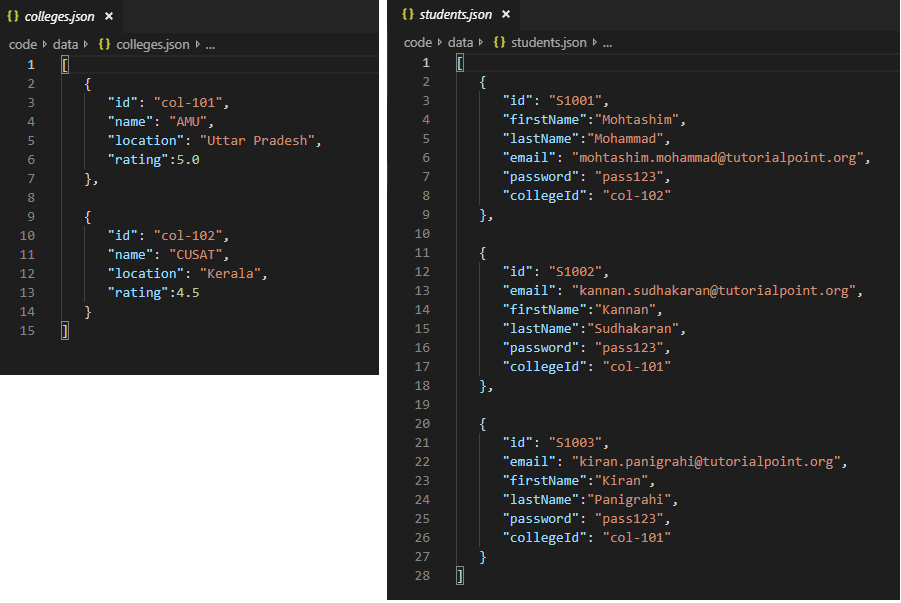


**Create Flat File Database in Data Folder**

we use flat files to store and retrieve data.

Create a folder data and add two files **students.json** and **colleges.json**.





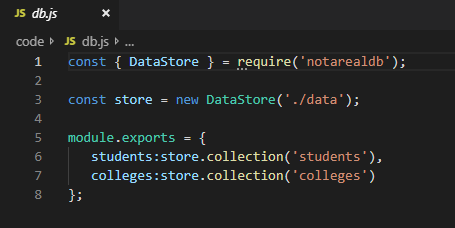
**Create a Data Access Layer**

We need to create a datastore that loads the data folder contents.

In this case, we need collection variables, *students* and *colleges*.

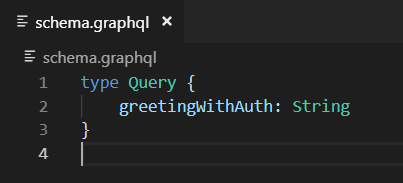
Whenever the application needs data, it makes use of these collection variables.

Create file db.js with in the project folder as follows



## Step 2 − Create a Schema

Add **schema.graphql** file in the project folder with following code

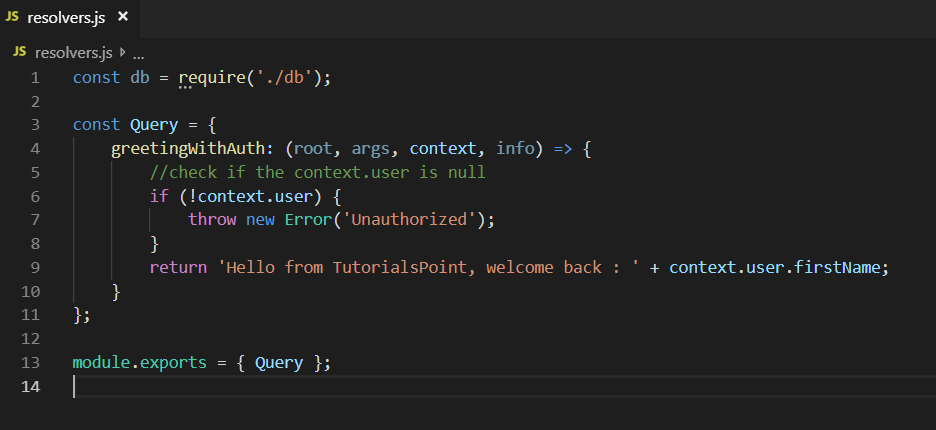


## Step 3 − Add Resolvers

Create a file **resolvers.js** in the project folder with following code

The resolver will verify if an authenticated user object is available in the context object of GraphQL.

It will raise an exception if an authenticated user is not available.



## Step 4 − Create Server.js File

The authentication middleware authenticates callers using a JSON Web Token.

The URL for authentication is **http://localhost:9000/login**.

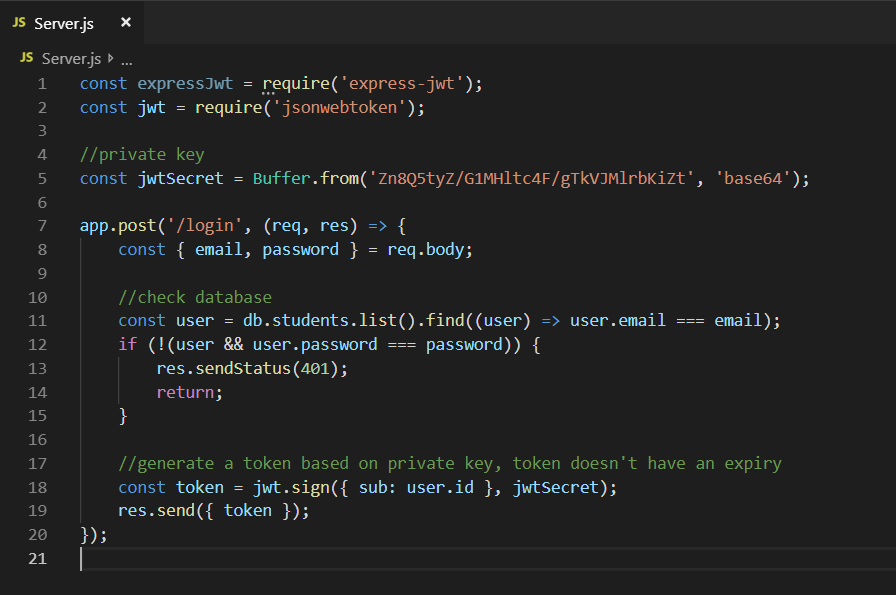
This is a post operation. The user has to submit his email and password which will be validated from the backend.

If a valid token is generated using jwt.sign method, the client will have to send this in header for subsequent requests.

If the token is valid, req.user will be set with the JSON object decoded to be used by later middleware for authorization and access control.

The following code uses two modules − **jsonwebtoken** and **express-jwt** to authenticate requests

* When the user clicks on the **greet** button, a request for the /graphql route is issued. If the user is not authenticated, he will be prompted to authenticate himself.
* The user is presented with a form that accepts email id and password. In our example, the /login route is responsible for authenticating the user.
* The /login route verifies if a match is found in the database for credentials provided by the user.
* If the credentials are invalid, a HTTP 401 exception is returned to the user.
* If the credentials are valid, a token is generated by the server. This token is sent as a part of response to the user. This is done by the jwt.sign function.



For every request, the app.use() function will be called.

This in turn will invoke the **expressJWT middleware**.

This middleware will decode the JSON Web Token. The user id stored in the token will be retrieved and stored as a property user in the request object.

//decodes the JWT and stores in request object

app.use(expressJwt({

secret: jwtSecret,

credentialsRequired: false

}));

To make available the user property within GraphQL context, this property is assigned to the **context** object as shown below

//Make req.user available to GraphQL context

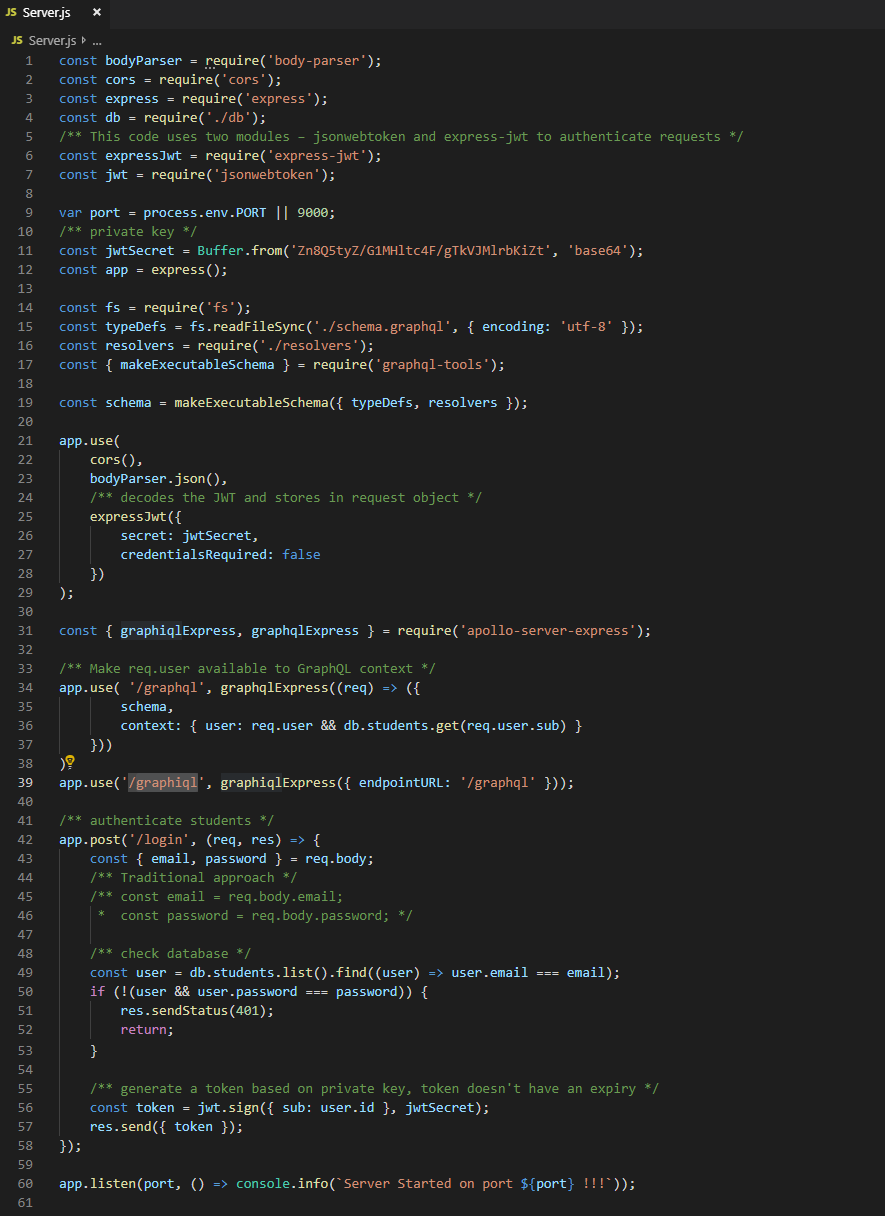
app.use('/graphql', graphqlExpress((req) => ({

schema,

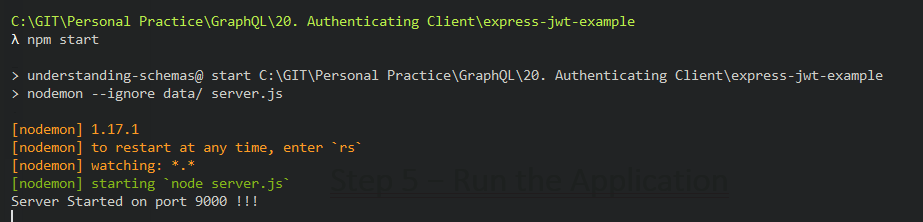
context: {user: req.user &&apm; db.students.get(req.user.sub)}

})));

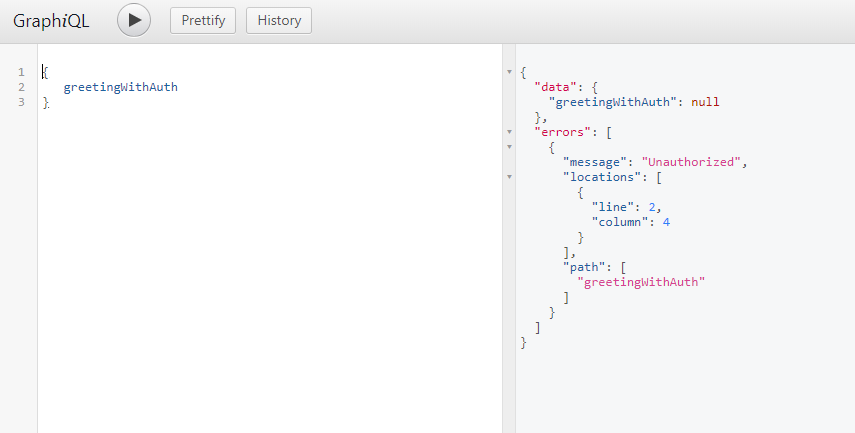
Complete Server.js looks like



## Step 5 − Run the Application



we got an error as **we are not authenticated user**



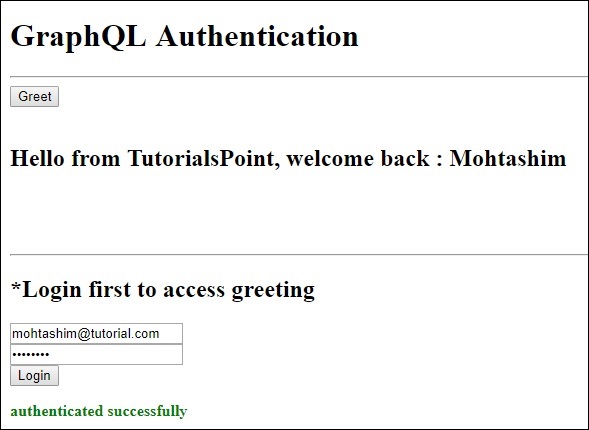
# Setting up the JQuery Client

In the client application, a greet button is provided which will invoke the schema **greetingWithAuth**.

If you click the button without login, it will give you the error message as below



Once you log in with a user available in database, the following screen will appear



To access **greeting**, we need to first access the URL **http://localhost:9000/login** route as below.

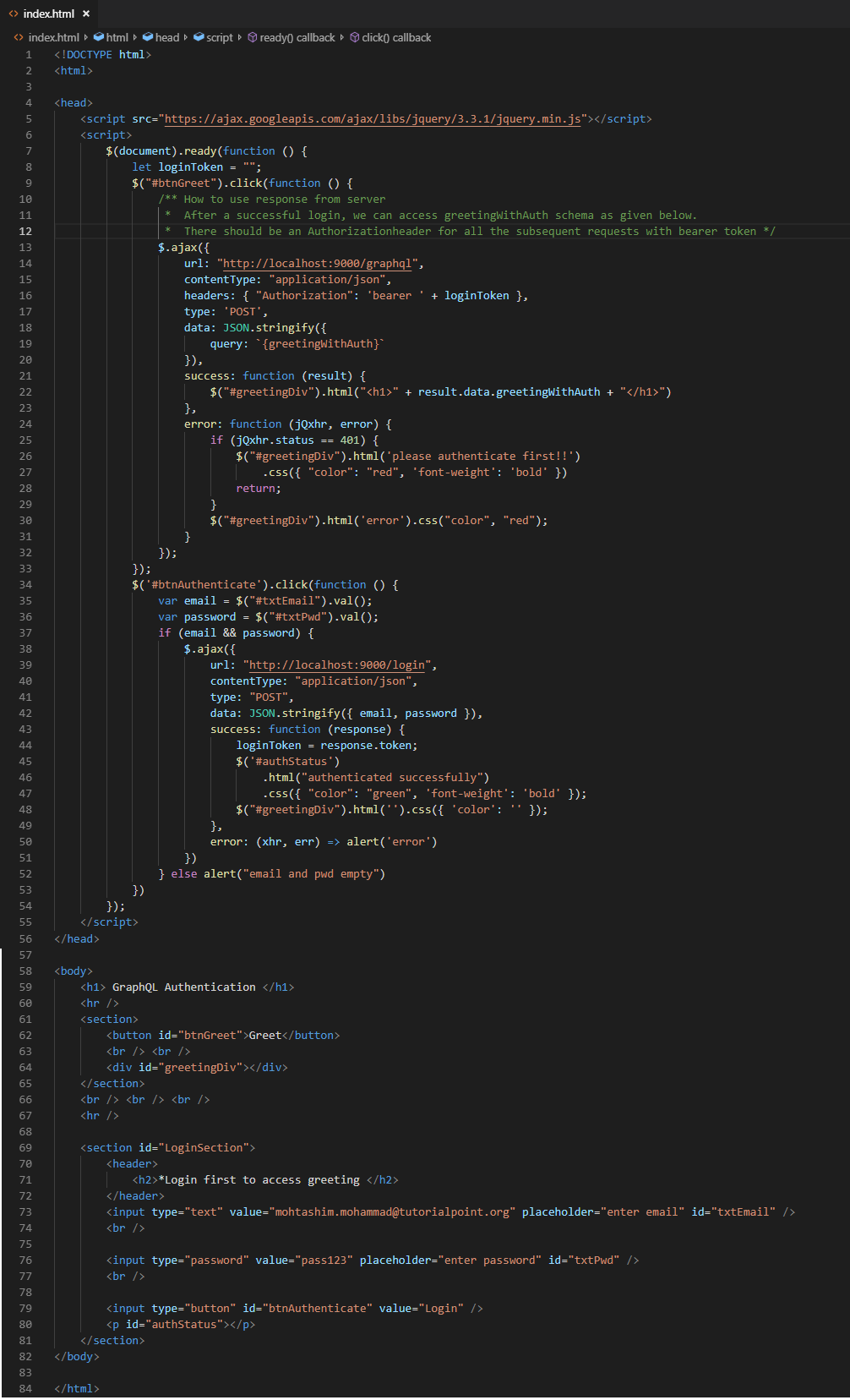
The response will contain the token generated from the server.

After a successful login, we can access *greetingWithAuth* schema as given below.

There should be an Authorizationheader for all the subsequent requests with bearer token.

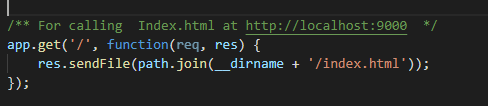


The following is the code for index.html

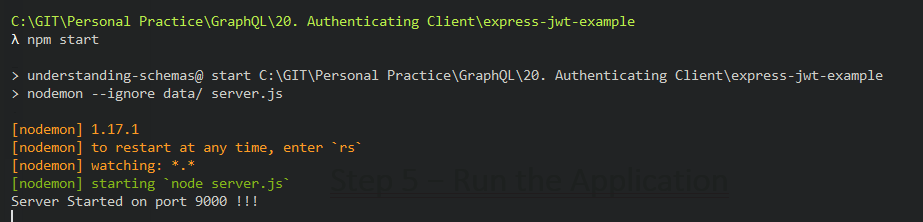


## Step 4 – Update required for Server.js

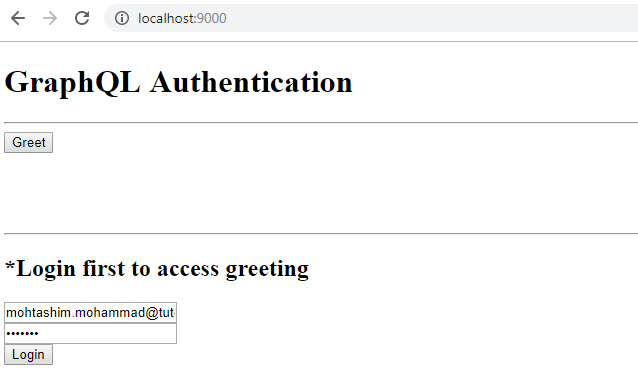
In order to invoke index.html via express, we need to add below code in server.js



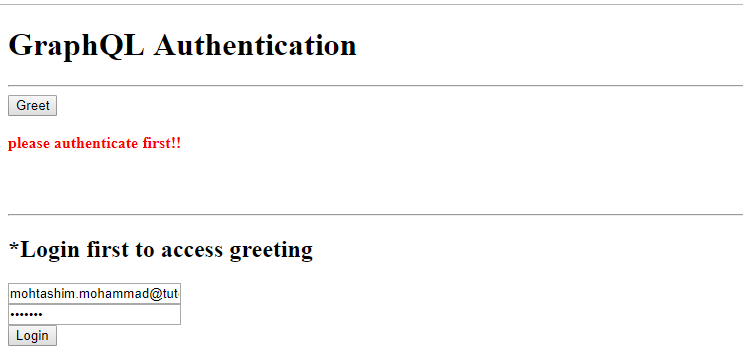
## Step 5 − Run the Application



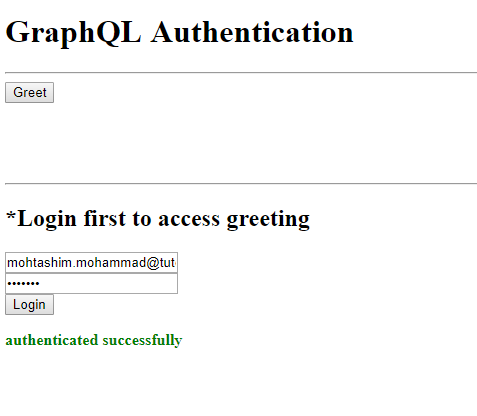
On Page Load



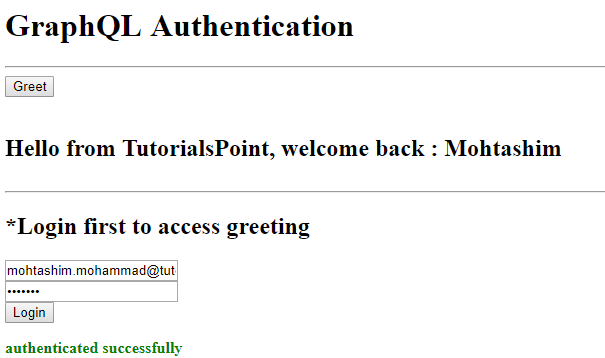
On clicking Greet button



On Clicking Login button



On Clicking Greet Button



<https://www.tutorialspoint.com/graphql/graphql_authenticating_client.htm>