Food Order App - Backend App with ExpressJS

Introduction

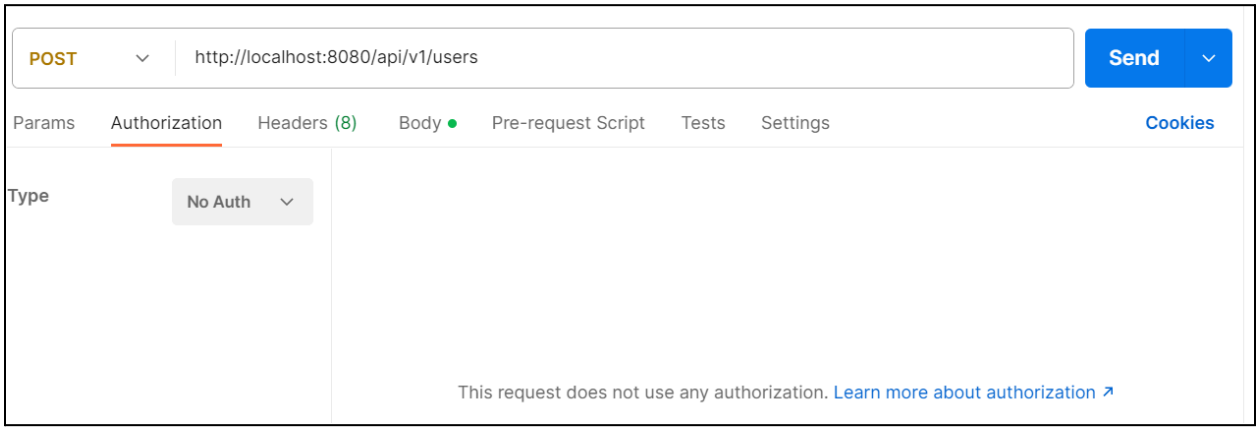
This week you will be building APIs for food items with ExpressJS for Food Order Application.

We have provided the starter code for the backend app which you will be working on.

Please refer to the **Food Order App_Backend Server_Design Document.pdf** file to understand the design of the database, the relationship among different files in the application and the working of APIs.

To run the Node Server App in the localhost kindly follow the below mentioned steps -

1. Download and extract the **Foodorder_app_backend** folder from the **ExpressJS_Assessment_For Noncoders_Source Code.zip** folder.
2. In the NoSQL week, you have already created a database named **food_order_app** using MongoDB Compass/Atlas. We will use the same database here.
3. Open the **constants.js** file from the app folder and update the value of **DATABASE_URI** variable with the connection string of your MongoDB database along with the database_name.
   For example: ***mongodb+srv://[username:password@]host/[defaultauthdb]***
   The connection string can be retrieved from the MongoDB compass or Atlas.
   Replace **defaultauthdb** with food_order_app (or the name of the database you have created).
4. Open a terminal and navigate to the **Foodorder_app_backend** folder in it.
5. Run the "*npm install*" command to install all the dependencies.
6. Now run the *"npm start"* command to run the application in the localhost. This will run your application on localhost on port 8080.
7. Now open the Postman tool to connect to the APIs of the backend application. The base URL for the node server is http://localhost:8080/api/v1/
8. To run your application there has to be a registered user in the database , for that you can follow the given steps -
   a. Make sure Auth is "No Auth" selected as Authorization header in Postman.



b. Create a new user using POST request URL http://localhost:8080/api/v1/users and body in JSON format as -

{

"username":"john35@d",

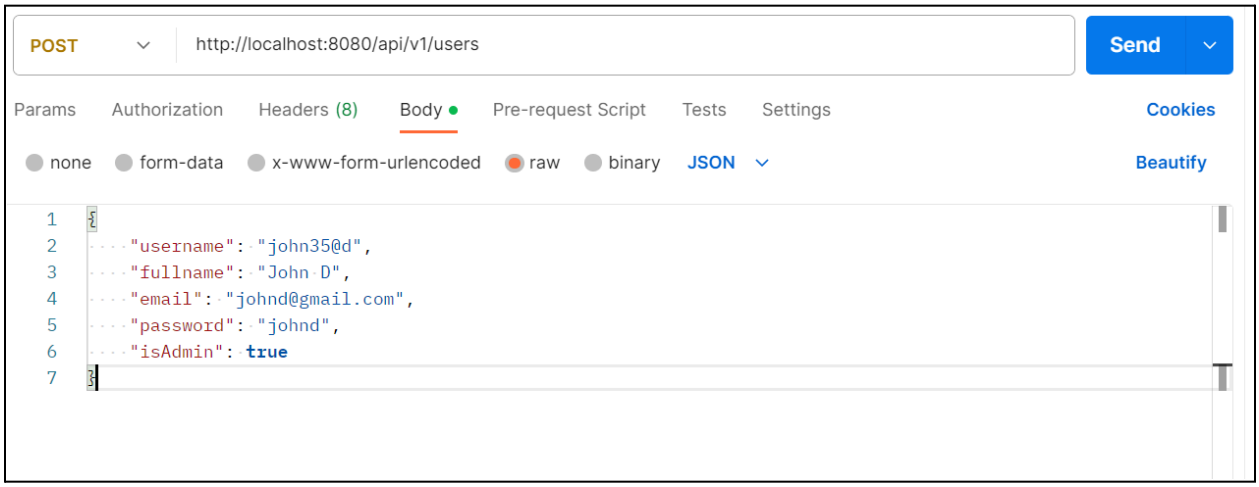
"fullname" : "John D",

"email" : "johnd@gmail.com",

"password" : "johnd",

"isAdmin" : true

}

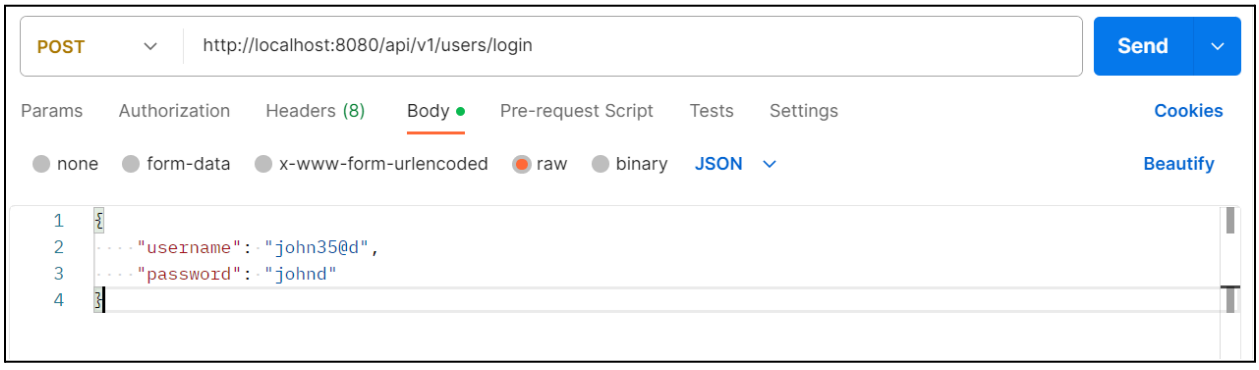


c. Login using POST request URL http://localhost:8080/api/v1/users/login and body in JSON format as -{

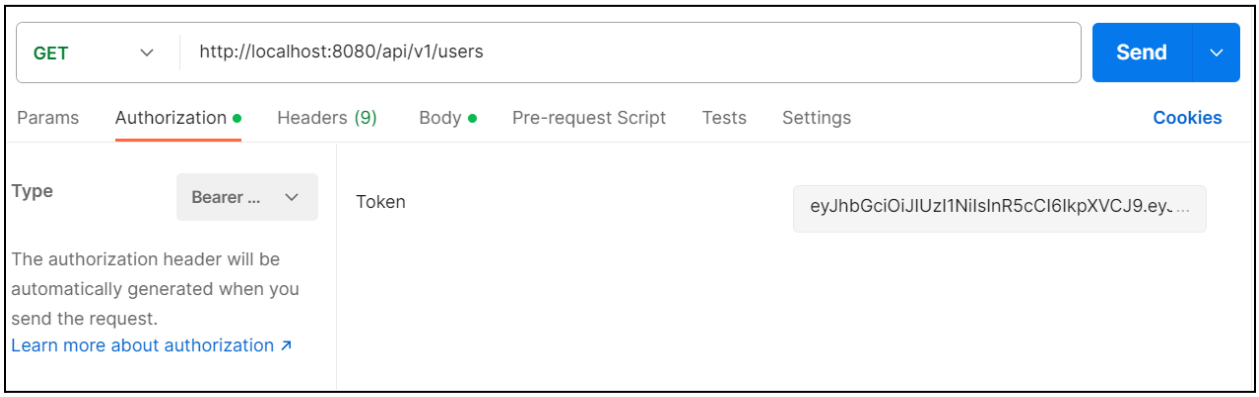
"username":"john35@d",

"password" : "johnd"

}



d. Copy the session token generated for the logged in user in **usersessionmodels** collection from MongoDB database and paste in the **Authorization->Bearer Token**.



e. Give any POST or GET request like if you want to fetch all categories you will give GET URL as http://localhost:8080/api/v1/categories and if you want to fetch specific category you will give its id in the query parameter for eg- http://localhost:8080/api/v1/categories/categoryId will fetch category with the given category Id.

Housekeeping points

- This is a minimal example and may not follow some standard practices.
- We focus on the main flow, and not much error handling.

Problem Statement

Your task is as follows -

1. **Implementation of APIs for fooditems (File name - fooditemService.js URL Example - http://localhost:8080/api/v1/fooditems)**
   a. **Complete the code of `createFooditem` –** It should use the createFooditem() of fooditemRepository.js file to create a new fooditem by validating the data provided in the HTTP request and should send appropriate HTTP response.
   b. **Complete the code of `editFooditem` –** It should use the `editFooditem`() of fooditemRepository.js file to edit existing fooditem by validating the data provided in the HTTP request and should send appropriate HTTP response.
   c. **Complete the code of `deleteFooditem` –** It should use the `deleteFooditem`() of fooditemRepository.js file to delete the existing fooditem by validating the data provided in the HTTP request and should send appropriate HTTP response
   d. **Complete the code of `getFooditem` –** It should use the `getFooditem`() of fooditemRepository.js file to return the existing fooditem requested as per the id provided by validating the data provided in the HTTP request and should send appropriate HTTP response.
   e. **Complete the code of `getAllFooditems` –** It should use the `gettAllFooditems`() of fooditemRepository.js file to return the list of all existing fooditems and should send appropriate HTTP responses.

Program Organization

- You will be getting a folder named **ExpressJS_Assessment_For Noncoders_Source Code** having a sub-folder **Foodorder_app_backend** further having a subfolder named **app** and a file named **server.js** file.

Evaluation Rubric

Total Project Points: 60

- Correctness:
  Correctness of implementation
  - Problem statement - point a (**20%**) : **12 Points**
  - Problem statement - point b (**20%**) : **12 Points**
  - Problem statement - point c (**20%**) : **12 Points**
  - Problem statement - point d (**20%**) : **12 Points**
  - Problem statement - point e (**20%**) : **12 Points**

Program Instructions

- Make sure to remove the node_modules folder before submitting the project.
- Make sure you zip the **Foodorder_app_backend** folder before submitting the project.
- Project will not be evaluated if the submitted project is not in the zip/rar format.