

Instructions:

N.B. The database used in this project is mongodb - one can either use the in memory mongodb instance that is automatically provided and connected to by the api code or provide an URI to a mongodb database of choice by overriding the environment variable:

MONGO_DB_URI

Run the following commands in the root directory to start the server:

1. `npm ci`
2. `npm run start:dev`

Output such as the following should appear in the console:

```
added 398 packages in 63.131s
~/Dropbox/career/coding_challenge/monaco/monaco-api > npm run start:dev

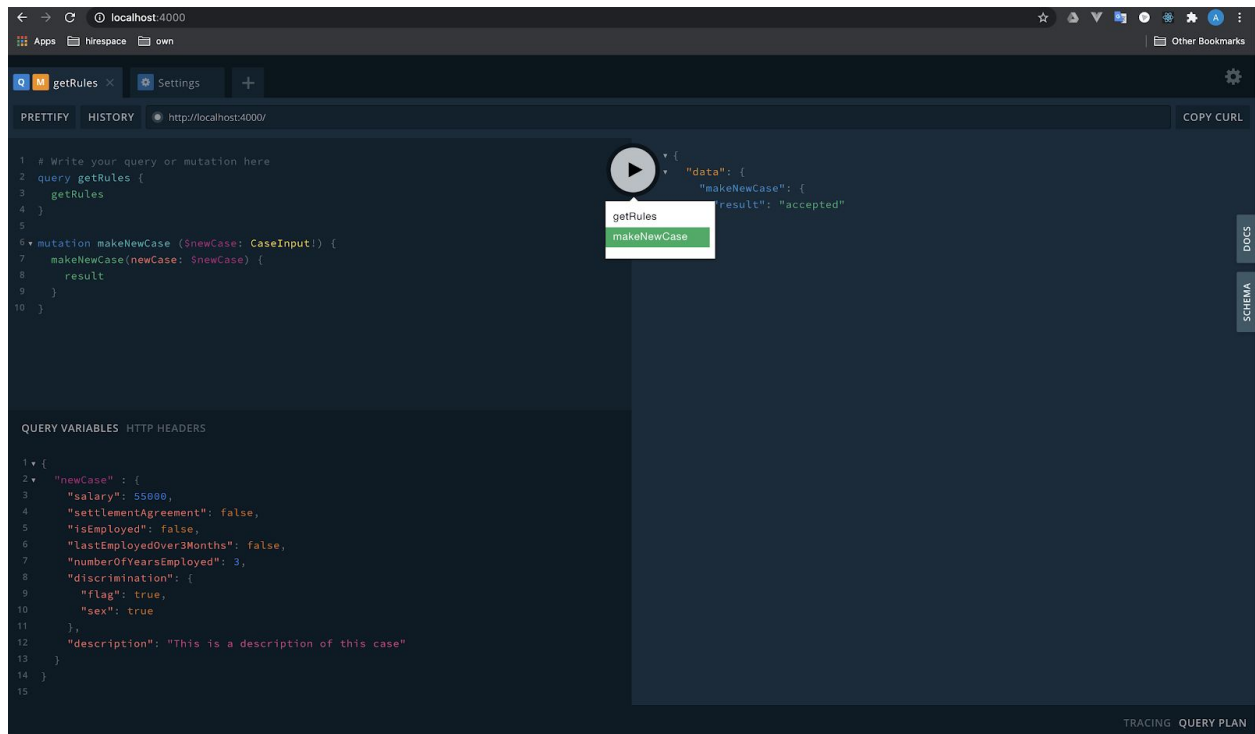
> monaco-api@1.0.0 start:dev /Users/marwat/Dropbox/career/coding_challenge/monaco/monaco-api
> nodemon

[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src/**/*
[nodemon] watching extensions: ts,js
[nodemon] starting `ts-node ./src/index.ts`
✖ Server ready at http://localhost:4000/
Started mongodb in memory database with dbName: ac9322b2-ac68-4d4b-beb9-4122fd69ca1c
Started mongodb in memory database with port: 62808
Started mongodb in memory database with dbPath: /var/folders/0m/51ypwz6s0jzdwzjm7_0lv_6r0000gn/T/mongo-mem--87224-7LRJXzm0LZ67
Started mongodb in memory database with uri: mongodb://127.0.0.1:62808/ac9322b2-ac68-4d4b-beb9-4122fd69ca1c?
```

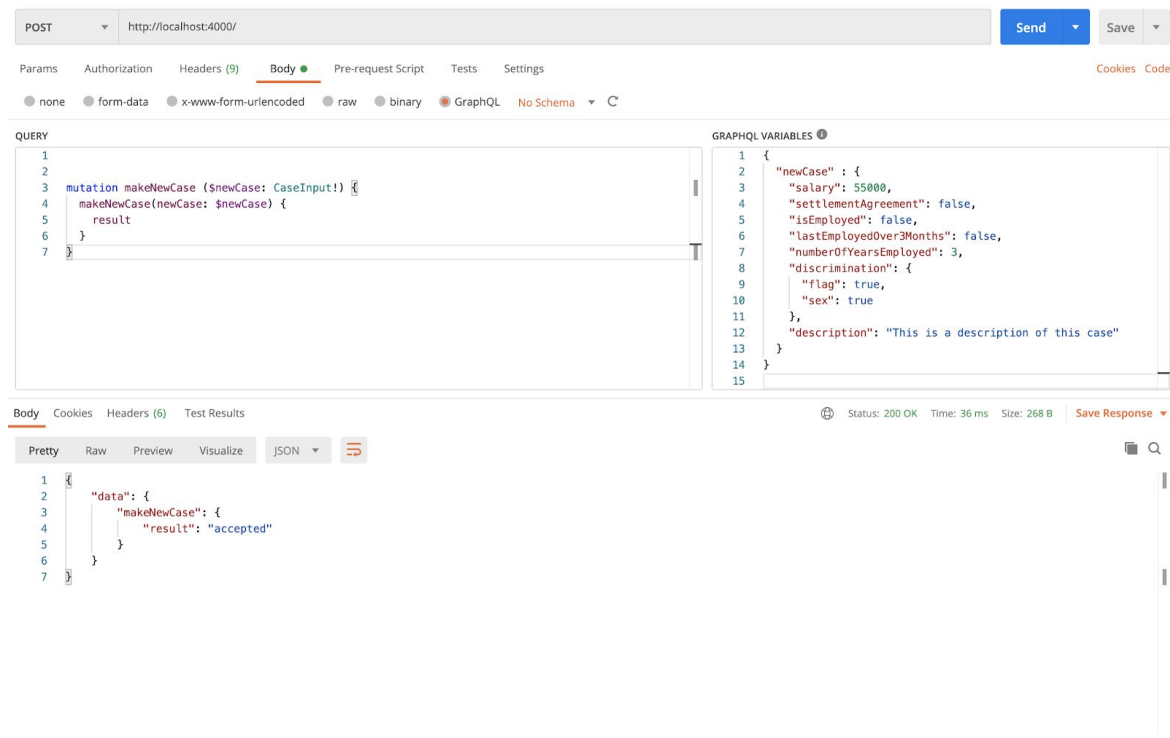
Where <http://localhost:4000/> is the API endpoint, powered by GraphQL. The details below are that of the in memory database used to store 'case' data.

2. To reach the API you can either click on the link in the console: <http://localhost:4000/> and the GraphQL playground should appear or you could also use tools like Postman (Post and GraphQL option in Body), curl, etc. The playground has the added benefit of giving an overview of the schema for the API and all the calls that can be made. Example snippets are provided below:

GraphQL playground directly opened by clicking on link <http://localhost:4000/> :



Postman:



The queries for 1) getting rules and 2) making a new 'case', and also the associated variables that are used for input are stored in the test-data folder.

Rules:

As requested the rules are separately stored in the config folder. They can be used to alter the business related logic for accepting or rejecting a case without having to make any code changes:

```
import {Discrimination} from '../data/types';

export const config = {
  rules: {
    conditions: {
      any: [{
        all: [{
          fact: 'case',
          operator: 'greaterThan',
          value: 100000,
          path: '.salary'
        }, {
          fact: 'case',
          operator: 'greaterThan',
          value: 2,
          path: '.numberOfYearsEmployed'
        }]
      }, {
        all: [{
          fact: 'case',
          operator: 'equal',
          value: true,
          path: '.settlementAgreement'
        }]
      }, {
        all: [{
          fact: 'case',
          operator: 'greaterThan',
          value: 30000,
          path: '.salary'
        }, {
```

This functionality was implemented using the help of the following library:

<https://github.com/CacheControl/json-rules-engine>

Database:

To see the results in the mongodb database one can connect to the database using the uri output in the console and a tool such as Robo 3T :

Key	Value	Type
(1) ObjectId("5f35a6dc8a71f454b92d665e")	{ 9 fields }	Object
(2) ObjectId("5f35a7e38a71f454b92d665f")	{ 9 fields }	Object
(3) ObjectId("5f35a7e58a71f454b92d6660")	{ 9 fields }	Object
_id	ObjectId("5f35a7e58a71f454b92d6660")	ObjectId
salary	55000	Int32
settlementAgreement	false	Boolean
isEmployed	false	Boolean
lastEmployedOver3Months	false	Boolean
numberOfYearsEmployed	3	Int32
discrimination	{ 2 fields }	Object
description	This is a description of this case	String
status	accepted	String
(4) ObjectId("5f35a9538a71f454b92d6661")	{ 9 fields }	Object
_id	ObjectId("5f35a9538a71f454b92d6661")	ObjectId
salary	1000	Int32
settlementAgreement	false	Boolean
isEmployed	false	Boolean
lastEmployedOver3Months	true	Boolean
numberOfYearsEmployed	3	Int32
discrimination	{ 2 fields }	Object
description	This is a description of this case	String
status	rejected	String

How To Improve:

More understanding of business requirements would help on how to better design and improve this API, that said, some steps I can think of are:

1. Abstract the rule checking process into a separate module
2. Provide an interface for business users to amend the rules
3. Add better error handling
4. Expand the API to provide more information as to why a case was rejected or accepted
5. Add testing, both at a unit level and at a functional level across the whole API