Node.js – Express + GraphQL Event Management API Documentation

Project requirements:

- Node.js Runtime
- MongoDB
- Node Package Manage (NPM)

Configuration Changes:

If you wish to modify the running port of server or database connection string you may do it in base_dir/server.config.js

GraphiQL

Is a tool for development and testing, that provides the GUI interface for browsers when accessing a graphql end-point by providing useful features like docs, prettier, query intellisense.

Should be disable in production, 2 methods of doing that

1. change the environment to production in the server.config.js file.
2. change value of graphiql to false on line 57 of server.js file.

Setup:

- Go to the base directory
- Open terminal
- Run the command “npm install” to install all dependencies
- Run the command “npm run dev” to start the server in development mode.
- Or run “node server.js“ to run the server in production mode

End points

The Back-end API serve features only 1 endpoint (serving as a root ending for all queries) that supports both GET and POST requests

Endpoint URL “serverbaseurl:PORT/graphql” example “localhost:5000/graphql”

Which can be changed by editing it on the line 55, inside server.js.

GraphQL

Official Doc: https://graphql.org/learn/

Tutorial on YouTube: https://youtu.be/ZQL7tL2S0oQ

Queries And Mutation

Queries – refers to the data fetching process of the GraphQL server. (i.e., SELECT in SQL, GET in Rest APIs).

Mutation – refers to all the data modification process, Insert/Update/Delete.

Query – To do any kind of operation we need to send query to the server (can be a query/mutation) similar to SQL in MySQL database.

To check all available queries, you can go to the http://localhost:5000/graphql and check the docs section at top right or keep reading.

General Syntax:

```
query {
        typeofquery (…arguments) {
                …fields to be fetched
        }
}
```

OR

```
mutation {
        typeofmutation (…arguments) {
                …fields to be fetched
        }
}
```

Sending HTTP Request

Useful links:

https://graphql.org/learn/serving-over-http/

https://www.youtube.com/watch?v=0ZJI4cBS4JM&t=601s

all the queries need to be passed in http body

simple analogy for POST request

query	http (body)

query {
admins (username: "Admin", password: "Admin") {
admin_username
}
}
	{
	"query": "query { admins(username: \"Admin\", password: \"Admin\") { admin_username } }"
	}

For GET request one may do:

query	Get URL

{
me {
name
}
}
	http://myapi/graphql?query={me{name}}

**make sure to URL encode your string

Response:

Regardless of the method by which the query and variables were sent, the response should be returned in the body of the request in JSON format. As mentioned in the spec, a query might result in some data and some errors, and those should be returned in a JSON object of the form:

{

"data": { ... },

"errors": [...]

}

Example queries for specified end points

**Note words that colored red are variable inputs that must be passed accordingly.

Also note that some argument for some particular query maybe compulsory, check it using the GraphiQL docs, indicatated by a "!" mark at the end of its input type, example: username: String!

1. /admin_login

   For checking login credentials

   Query form:

```
query {
        admins (username: "Admin", password: "Admin") {

                admin_username
        }
}
```

   GET form (not recommended):

   http://localhost:5000/graphql?query=query%20%7B%0A%20%20admins%20(username%3A%20%22Admin%22%2C%20password%3A%20%22Admin%22)%20%7B%0A%20%20%20%20admin_username%0A%20%20%7D%0A%7D

   POST form (recommended) (inside http body):

```
{
        "query": "query {
                admins(username: \"Admin\", password: \"Admin\") {
                        admin_username
                 }
        }"
}
```

2. /contact_insert

   Adding a contact message by the user.

   Query Form
   **Note all field inside { } are not mandatory but at least one should be queried.

```
mutation {
 insertContact(
   contact_name: String!
   contact_emailid: String!
   contact_phoneno: String!
   contact_message: String!
   contact_business_name: String!
   contact_business_type: String!
 ) {
```

```
    _id,
    contact_name,
    contact_emailid,
    contact_phoneno,
    contact_message,
    contact_business_name,
    contact_business_type,
  }
}
```

Post Form:

```
{
  "query": "mutation {
    insertContact(
    contact_name: String
    contact_emailid: String
    contact_phoneno: String
    contact_message: String
    contact_business_name: String
    contact_business_type: String
  ) {
      _id,
      contact_name,
      contact_emailid,
      contact_phoneno,
      contact_message,
      contact_business_name,
      contact_business_type
    }
  }"
}
```

3. /contact_display

Displays all the contact messages

Query Form

```
query {
  contacts {
    _id,
    contact_name,
    contact_emailid,
    contact_phoneno,
    contact_message,
    contact_business_name,
    contact_business_type,
  }
}
```

POST Form / For GET Form convert the query into URL encoded string and pass it as query para in the URL demo: "localhost:5000.graphql/query=querystring.

```
{
  "query" : "query {
```

```
 contacts {
    _id,
    contact_name,
    contact_emailid,
    contact_phoneno,
    contact_message,
    contact_business_name,
    contact_business_type,
  }
 } “
}
```

4. /event_insert

Adds an event to the DB

Query Form

```
mutation {
 insertEvent(e_title: "String", e_sub_title: "String", e_about_title: "String", e_about_text:
"String", e_date: "String", e_time: "String", e_venue: "String", e_venue_link: "String",
e_speaker_one: "String", e_speaker_two: "String", e_speaker_three: "String",
e_speaker_one_photo: "String", e_speaker_two_photo: "String", e_speaker_three_photo:
"String", e_speaker_one_designation: "String", e_speaker_two_designation: "String",
e_speaker_three_designation: "String", e_status: "String", template_id: 1) {
   _id
   e_title
   e_sub_title
   e_about_title
   e_about_text
   e_date
   e_time
   e_venue
   e_venue_link
   e_speaker_one
   e_speaker_two
   e_speaker_three
   e_speaker_one_photo
   e_speaker_two_photo
   e_speaker_three_photo
   e_speaker_one_designation
   e_speaker_two_designation
   e_speaker_three_designation
   e_status
   template_id
 }
}
```

POST Form

```
{
 “query” : “mutation {
 insertEvent(e_title: \"String\", e_sub_title: \"String\", e_about_title: \"String\",
e_about_text: \"String\", e_date: \"String\", e_time: \"String\", e_venue: \"String\",
e_venue_link: \"String\", e_speaker_one: \"String\", e_speaker_two: \"String\",
e_speaker_three: \"String\", e_speaker_one_photo: \"String\", e_speaker_two_photo:
```

```
\"String\", e_speaker_three_photo: \"String\", e_speaker_one_designation: \"String\",
e_speaker_two_designation: \" String\", e_speaker_three_designation: \"String\",
e_status: \"String\", template_id: 1) {
    _id
    e_title
    e_sub_title
    e_about_title
    e_about_text
    e_date
    e_time
    e_venue
    e_venue_link
    e_speaker_one
    e_speaker_two
    e_speaker_three
    e_speaker_one_photo
    e_speaker_two_photo
    e_speaker_three_photo
    e_speaker_one_designation
    e_speaker_two_designation
     e_speaker_three_designation
    e_status
    template_id
   }
 } “
}
```

5. /event_display

   Gets the events with **e_status: “active”**.

Query Form

```
{
 events {
   _id,
   e_title,
   e_sub_title,
   e_about_title,
   e_about_text,
   e_date,
   e_time,
   e_venue,
   e_venue_link,
   e_speaker_one,
   e_speaker_two,
   e_speaker_three,
   e_speaker_one_photo,
   e_speaker_two_photo,
   e_speaker_three_photo,
   e_speaker_one_designation,
   e_speaker_two_designation,
   e_speaker_three_designation,
   e_status,
   template_id
```

```
 }
}
```

POST Form

```
{
 "query": "query {
  events {
    _id
    e_title
    e_sub_title
    e_about_title
    e_about_text
    e_date
    e_time
    e_venue
    e_venue_link
    e_speaker_one
    e_speaker_two: String
    e_speaker_three: String
    e_speaker_one_photo
    e_speaker_two_photo: String
    e_speaker_three_photo: String
    e_speaker_one_designation
    e_speaker_two_designation: String
    e_speaker_three_designation: String
    e_status
    template_id
  }
 }"
}
```

6. /event_edit

Takes multiple field (must be the same as event schema) to update the specified field in the arguments, **_id is must**.

Query Form

```
mutation {
 updateEvent(
  _id: "id",
  e_title: "String",
  e_sub_title: "String",
  e_about_title: "String",
  e_about_text: "String",
  e_date: "String",
  e_time: "String",
  e_venue: "String",
  e_venue_link: "String",
  e_speaker_one: "String",
  e_speaker_two: "String",
  e_speaker_three: "String",
  e_speaker_one_photo: "String",
  e_speaker_two_photo: "String",
  e_speaker_three_photo: "String",
```

```
  e_speaker_one_designation: "String",
  e_speaker_two_designation: "String",
  e_speaker_three_designation: "String",
  e_status: "String",
  template_id: Int
) {
  _id,
  e_title,
  e_sub_title,
  e_about_title,
  e_about_text,
  e_date,
  e_time,
  e_venue,
  e_venue_link,
  e_speaker_one,
  e_speaker_two: String
  e_speaker_three: String
  e_speaker_one_photo,
  e_speaker_two_photo: String
  e_speaker_three_photo: String
  e_speaker_one_designation,
  e_speaker_two_designation: String
  e_speaker_three_designation: String
  e_status,
  template_id
 }
}
```

POST Form

```
{
  "query": "mutation {
    updateEvent(
      _id: \"id\",
      e_title: \"String\",
      e_sub_title: \"String\",
      e_about_title: \"String\",
      e_about_text: \"String\",
      e_date: \"String\",
      e_time: \"String\",
      e_venue: \"String\",
      e_venue_link: \"String\",
      e_speaker_one: \"String\",
      e_speaker_two: \"String\",
      e_speaker_three: \"String\",
      e_speaker_one_photo: \"String\",
      e_speaker_two_photo: \"String\",
      e_speaker_three_photo: \"String\",
      e_speaker_one_designation: \"String\",
      e_speaker_two_designation: \"String\",
      e_speaker_three_designation: \"String\",
      e_status: \"String\",
      template_id: Int
) {
       _id,
      e_title,
```

```
    e_sub_title,
    e_about_title,
    e_about_text,
    e_date,
    e_time,
    e_venue,
    e_venue_link,
    e_speaker_one,
    e_speaker_two,
    e_speaker_three,
    e_speaker_one_photo,
    e_speaker_two_photo,
    e_speaker_three_photo,
    e_speaker_one_designation,
    e_speaker_two_designation,
    e_speaker_three_designation,
    e_status,
    template_id
  }
 }"
}
```

7. /event_reg_user_insert

Registers a user for a particular event, takes event_id and user details.

Datetime is taken automatically by the system on successful registration.

Responses:

If already registered

```
{
      _id: "alreadyREG",
      event_id: "alreadyREG",
      user_id: "alreadyREG",
      datetime: "alreadyREG",
}
```

If event doesn't exist

```
{
     _id: "event404",
     event_id: "event404",
     user_id: "event404",
     datetime: "event404",
}
```

Else on successful registration the inserted row (_id, event_id, user_id, datatime) is returned.

Query Form

```
mutation {
  registerUserEvent(event_id: "id", user_name: "String", business_name: "String",
business_type: "String", contact_num: "String", user_emailid: "String",
user_address: "String", user_city: "String") {
    _id,
    event_id,
    user_id,
    datetime
  }
}
```

POST Form

```
{
    "query": "mutation {
  registerUserEvent(event_id: "id", user_name: "String", business_name: "String",
business_type: "String", contact_num: "String", user_emailid: "String",
user_address: "String", user_city: "String") {
      _id,
      event_id,
      user_id,
      datetime
    }
  }"
}
```

8. /user_display

Displays details of the user.

Note, field QA will return an array of questions and answers filled by the user.

Query Form

```
query {
            users {
      _id,
      user_name,
      business_name,
      business_type,
      contact_num,
      user_emailid,
      user_address,
      user_city,
      QA {
        question,
        answer
      }
    }
}
```

POST Form

```
{
   “query”: “query {
            users {
      _id,
      user_name,
      business_name,
      business_type,
      contact_num,
      user_emailid,
      user_address,
      user_city,
      QA  {
        question,
        answer
      }
    }
  }”
}
```

9. /other_question_insert

Insert a question into the DB

Query Form

```
mutation {
 otherQuestionInsert (question:"String"){
   _id, question
 }
}
```

POST form

```
{
    “query”: “mutation {
 otherQuestionInsert (question:"String"){
   _id, question
 }
}”
}
```

10. /user_answer_insert

Insert an answer to a question by the user

Query Form

```
mutation {
 insertAnswer (oq_id: "id", user_id:"id", answer: "string"){
   _id, oq_id, user_id, answer
 }
}
```

POST Form

```
{
   “query”: “mutation {
  insertAnswer (oq_id: "id", user_id:"id", answer: "string"){
    _id, oq_id, user_id, answer
  }
 }”
}
```

Model

Models in this project are defined in the base_dir/models/ folder

Total 7 models

Admin.js

const Admin = new GraphQLObjectType({
  name: "Admins",
  description: "Admin Table",
  fields: () => ({
  _id: {
  type: GraphQLNonNull(GraphQLID),
  },
  admin_username: {
  type: GraphQLNonNull(GraphQLString),
  }
  }),
});
	Define an object that can be queried, similar to a table.
	Have 2 fetchable fields:
	_id, primary key
	admin_username, NOT NULL STRING
	
	in DB another field admin_password also exist but is not fetchable for security reasons.
	
	**this is a schema not a querytype

const AdminType = {
  type: Admin,
  description: "Admin",
  args: {
  username: { type: GraphQLNonNull(GraphQLString) },
  password: { type: GraphQLNonNull(GraphQLString) }
  },
  resolve: async (parent, args) => {
  let admin_col = await loadDataBase();
  return admin_col.findOne({admin_username: args.username, admin_password: args.password});
  },
};
	Defines a query of GraphQL.
	**but it is only a object defining the mechanism of query, not the name of the query.
	
	Args: are the argument that maybe passed while writing the query, GraphQLNonNull() indicates that the following argument must be passed.
	
	Resolve: function
	Is a arrow function that is called when the query is fired, takes two parameters: parent, args.
	Where parent is the parent query object that is calling this query.
	Example tree of order of calling
	Root query -> Admin

Contact.js

Most of the code is similar to the admin Model, but here an additional mutationtype is defined call “insertContact” that allows to insert a contact object into DB.

const insertContact = {
  type: Contact,
  description: "Insert a contact",
  args: {
  contact_name: { type: GraphQLNonNull(GraphQLString) },
  contact_emailid: { type: GraphQLNonNull(GraphQLString) },
  contact_phoneno: { type: GraphQLNonNull(GraphQLString) },
  contact_message: { type: GraphQLNonNull(GraphQLString) },
  contact_business_name: { type: GraphQLNonNull(GraphQLString) },
  contact_business_type: { type: GraphQLNonNull(GraphQLString) },
  },
  resolve: async (parent, args) => {
  let contact_col = await loadDataBase();
  let resp = await contact_col.insertOne(args);
  const result = {...args, _id: resp.insertedId};
  return result;
  }
};
	Take 6 arguments equal to the column(fields) in the DB document(table).
	
	Resolve function here instructs the mongodb driver to insert the document accordingly.

All the other models are working in the same pattern.