**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

}

}"

}

1. **/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  }  }”  } |

1. **/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,  }  } “  } |

1. **/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  }  } “  } |

1. **/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  }  }”  } |

1. **/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  }  }”  } |

1. **/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  }  }” } |

1. **/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  }  }  }” } |

1. **/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  }  }” } |

1. **/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.