Calculator API
You need to build a Calculator API Contract application which performs the following operations:

- Addition
- Subtraction
- Multiplication
- Division

Before you start implementing the above operations, you need to check that your server is working fine. For that, perform GET request which lets you land on the home page successfully with the message Hello world!.

After that, you need to implement the following **POST** requests:
The input may contain positive, negative, and floating-point numbers.
Request body parameters for all endpoints:

- num1
- num2

1. **/add**
Response:
{
status: “success"|"failure"|"error”,
message: “the sum of given two numbers”,
sum: num1+num2
}

2. **/sub**
Response
{
status: “success"|"failure"|"error”,
message: “the difference of given two numbers”,
difference: num1-num2
}

3. **/multiply**
Response
{
status: “success"|"failure"|"error”,

message: “The product of given numbers”
result: num1*num2
}

4. **/divide**
Response
{
status: “success"|"failure"|"error”,
message: “The division of given numbers”
result: num1/num2
}

5. The status should be "error" in the following cases:
- num2 is 0 for endpoint /divide. In this case, the message should be "Cannot divide by zero"
- The request body parameters contain values less than -1M(10 lakhs), or the result is < -1M. In this case, the message should be "Underflow"
- The request body parameters contain values above 1M(10 lakhs), or the sum is > 1M. In this case, the message should be "Overflow"
- The request body parameters contain Strings. In this case, the message should be "Invalid data types"

Note: Please keep in mind you need to print messages as stated in the problem statement

School Administration
With online schooling on the rise, every school needs software to store the information of the students. New student records need to be added when admissions happen, their details updated when students pass to the next class and records need to be deleted when students leave

Details

1. A student record consist of following structure.
    1. id
    2. name
    3. currentClass
    4. division

2. The server should respond to GET, POST, PUT, DELETE request

GET request

1. /api/student
    1. Initial data is provided in InitialData.js file. This data should be preloaded in server. Like
       if first request to the server is get /api/student then it should return all these student
       records.

    2. Server should send back details of all student record available in the form of array in json
       format.

2. /api/student/{id}
    1. Server should send back details of provided student id in json format {id:1,name: 'Lokesh',
       currentClass : 12, division: 'A'}
    2. If id is invalid respond with 404.

POST request

1. Request will be sent with headers {'content-type':'application/x-www-form-urlencoded'}
2. /api/student
    1. Server should record the student details if all the three [name, currentClass, division] are
       provided. Should return the id to which is allocated to the new student details in json format {'id':new_id}
    2. If incomplete details are given return status code 400.
    3. Returned should be unique to every student and should be incremental.
    4. Even if record is deleted its id should not be used.

PUT Request

1. PUT request will have header {'content-type':'application/x-www-form-urlencoded'}
2. /api/student/{id}
    1. If valid id and valid update is given then should update the student details
    2. If invalid id is given then respond with status 400.

        3. If invalid update is given then respond with status 400.
        4. Update will be given with format {name: 'new name'}

DELETE request

1. /api/student/{id}
    1. If valid id is given remove the corresponding record.
    2. If invalid id is given respond with status 404.

Acceptance Criteria:

1. Status response should be as asked in each route.
2. Use of Proper Parser.
3. Response data should be in JSON format.

Get Youtube Subscribers

Get a List of Subscribers

You might not realize but databases are everywhere. Databases used to store records efficiently. We can do much more with database for example we can add, read, update and delete records form database. MongoDB, the most popular database for modern app, is a document-oriented NoSQL database used for high volume data storage.
Task: ***Create a simple Application using Mongodb in node.js***

Acceptance Criteria

- Do not start server and connect to database inside app.js
- app.js should only be used to handle request and response
- Use index.js or other file to connect and start server
- Run node src/createDatabase.js to create Database on local your local computer(Only works if you have MongoDB installed in local machine)
  GET http://localhost:3000/subscribers
  Response with an array of subscribers(an Object)

  GET http://localhost:3000/subscribers/names
  Response with an array of subscribers(an Object with only two fields name and subscribedChannel)

  GET http://localhost:3000/subscribers/:id
  1. Response with a subscriber*(an object)* with given id

2. Response with status code 400 and Error message({message: error.message}) if **id** does not match

Mario CRUD

You might not realize but databases are everywhere. Databases used to store records efficiently. We can do much more with database for example we can add, read, update and delete records form database. MongoDB, the most popular database for modern app, is a document-oriented NoSQL database used for high volume data storage.
Task: ***Create CRUD Application using Mongodb in node.js***

Acceptance Criteria

- Do not start server and connect to database inside app.js
- app.js should only be used to handle request and response
- Use index.js or other file to connect and start server
- All Responses should be in JSON format

- **Mario character Details**

    1. An Object with two properties *name(string) and weight(Number)*
    2. {name:"Luigi", weight: 60}
    3. Model marioModel
    4. Model Name mariochar

GET http://localhost:3000/mario

1. Response with an array of Mario characters.

GET http://localhost:3000/mario/:id

1. Response with a Mario character with given id
2. Response with status code 400 and Error message({message: error.message}) if **id** does not match

POST http://localhost:3000/mario

1. Save the given Mario character in the database. Response with status code 201 and newly saved Mario character
2. Response with status code 400 and Error message({message: 'either name or weight is missing'}) if character's **name or weight** is misssing

PATCH http://localhost:3000/mario/:id

1. Response with a Mario character of given id after updating as per suggested changes in database
2. Response with status code 400 and Error message({message: error.message}) if **id** does not match or invalid suggested changes

DELETE http://localhost:3000/mario/:id

1. Delete the Mario character of given id from the database. Response with status code 200 and message({message: 'character deleted'})
2. Response with status code 400 and Error message({message: error.message}) if **id** does not match

Api Rate Limiting

Rate limiting

Rate limiting protects your APIs from overuse by limiting how often each user can call the API. This protects them from inadvertent or malicious overuse. Without rate limiting, each user may request as often as they like, leading to "spikes" of requests that starve other consumers.
In case a client made too many requests within a given time frame, HTTP servers can respond with Status code: 429, Too Many Requests.

Some API Rate limiting techniques

- Caching
- Limit number of API calls
- Delay response time

Implement rate limiting in API

Create your own RESTful API.
GET http://localhost:3000/api/posts – This will get an array of 10 posts with status code 200
GET http://localhost:3000/api/posts?max=15 – This will get an array 15 posts with

status code 200
each post is an object, see initialData.js

Acceptance Criteria

If users make API call without specifying max then response with an array of 10 posts (default case) with status code 200

If user makes an API call then for 30sec if user makes another calls then respond with an array of posts where number of posts will be minimum of first API call's max and current API call's max. *max denotes number of post and default value is 10*

if user makes more than 5 API calls within 30sec then respond with {message: "Exceed Number of API Calls"} with status code 429 *After 30sec resume everything back to initial state*

Maximum number of posts should be 20 only if max is specified and less than or equal to 20 otherwise it's 10. *ex: http://localhost:3000/api/posts?max=21 should return an array of 10 posts only.*

Covid Statistics
During any virus outbreak, the governments are supposed to constantly look at the new hotspots, mortality rates, etc of different states to aid the states with necessary equipment and help. To do so, the officials must get correct and reliable statistics immediately.
We are supposed to create an API server that performs some aggregation tasks on the Covid data. This data is stored in the MongoDB server.
You can create this dataset on your desktop if you have MongoDB installed on it.

Run node /src/createDatabase.js . Read the file carefully to understand how it is working.

Don't forget to run npm i to install dependencies

- Acceptance Criteria

1. We are supposed to expose our server at 5 endpoints.
   1. **GET** **localhost:8080/totalRecovered**
      should calculate the total number of recovered patients across all the given states(also UTs).
      Returned response should be in the format {data: {_id: "total", recovered:

135481}}.
2. **GET** **localhost:8080/totalActive**
   should calculate the total number of active patients across all the given states(also UTs).
   Active cases = (infected-recovered)
   Returned response should be in the format {data: {_id: "total", active: 11574}}.
3. **GET** **localhost:8080/totalDeath**
   should calculate the total number of deaths across all the given states(also UTs).
   Returned response should be in the format {data: {_id:"total", death: 11574}}.
4. **GET** **localhost:8080/hotspotStates**
   Every state is declared as a hotspot of its rate value is greater than 0.1.
   rate value can be calculated as ((infected - recovered)/infected)
   rate value should be rounded to 5th decimal point using MonogDb built-in feature $round.

   Returned response should be in the format {data: [{state: "Maharastra", rate: 0.17854}, {state: "Punjab", rate: 0.15754}]}.
5. **GET** **localhost:8080/healthyStates**
   Every state is declared as a healthy state of its mortality value is less than 0.005.
   mortality value can be calculated as (death/infected).
   mortality value should be rounded to 5th decimal point using MonogDb built-in feature $round.

   Returned output should be in the format {data: [{state: "Maharastra", mortality: 0.0004}, {state: "Punjab", mortality: 0.0007}]}.

Note

1. Mongoose should be used.
2. Tests are Case Sensitive so be careful with output data
3. Output format should be as mentioned above.
4. Round off the data using built-in feature $round

Creating a News API server

In this fast-growing economy, it's very necessary to be up to date with the news. Easier said than done. One of the major problems is a slow news server. So let's do something about it. We can create our own api to send stored news data from our own MongoDB server.

This API also needs to be paginated. We should be able to define the offset and the limit in the API call through query parameters.

Details:

1. After npm install, you can create the news database on your system using command node ./src/createDatabase.js.
2. Schema, Model and connector are already provided for you to use.
3. We want to send the data stored in collection named dailynews
4. We have to expose only one endpoints at localhost:8080
5. **GET http:/locahost:8080/newFeeds**

    - Two parameters limit, offset can also be passed along with this request.
    - Parameter limit will define number of records that are required in response.
    - Parameter offset will define number of records that are suppose to skip from the first recorded in the collection.
    - When invalid value of parameters is passed then consider default value for it.
    - Default value of limit is 10 and for offset its 0.
    - If records are found for given limit and offset send it in the form of array without editing the record. Don't edit the structure.
    - If no records are found, return empty array.

Acceptance criteria

1. Use of MongoDB is compulsory.
2. Sending the data in the expected structure

School Leaderboard

Do you remember your ranking in the last competition held at your School? How did you check it? Was it a leaderboard? Leaderboards are an excellent way to test your progress and help you stay focused. Now that we know its importance, we should be able to create one ourselves. When we display scores on the

leaderboard, if the data is too big, we prefer to display it in a paginated fashion. Only some limited data is shown on the first screen to not make the website overcrowded.
Let's learn how paginated APIs are built and how the backend server can be able to support them.

Details

- File name data.js contains the details about top 160 coders.
- Expose an endpoint on http://localhost:8080

  1. Path /topRankings
  2. Method: GET
  3. Parameters:
     1. limit (Optional, Positive integer, Default Value = 20)
     2. offset (Optional, Positive integer, Default Value = 0)

  4. Response data: List of top coders in the format:
     Array<{ name: String, username: String, global_rank: Number, country_rank: Number, rating: String, diff: Number, country_code: String, country: String, institution: String, institution_type: String, all_rating: String, user_gender: String}>

- **Examples:**
- GET request on http://localhost:8080/topRankings?limit=5&offset=1: The server should skip the first record from data.js file and return next 5 records.
- GET request on http://localhost:8080/topRankings?limit=2&offset=0 [{ "name": "Gennady Korotkevich",
  "username": "gennady.korotkevich",
  "global_rank": 1,
  "country_rank": 1,
  "rating": 3923,
  "diff": 0,
  "country_code": "BY",
  "country": "Belarus",
  "institution": "Saint Petersburg National Research University of Information Technologies, Mechanics and Optics",
  "institution_type": "College",
  "all_rating": "3923",
  "user_gender": "Male" }, {
  "name": "Mun.HakMyong",
  "username": "rns4",

```
"global_rank": 2,
"country_rank": 1,
"rating": 3061, "diff": 0,
"country_code": "KP",
"country": "North Korea",
"institution": "Kim Il-sung University",
"institution_type": "College",
"all_rating": "3061",
"user_gender": "Male"
}]
```

Search College
Searching for a college could be very hard especially when we have a lot of choices. But it is very necessary to choose the right one and be aware of all the options. Now that we are in college and are familiar with what issues we all faced, we should try to help others. We can create a robust server that can handle some of their problems.
Details

1. Our server should search according to all the parameters specified by the user
2. The search should be case-insensitive.
3. Schema, Model and connector are already provided for you to use.
4. We want to send the data stored in a collection named collegerecords
5. Structure of the data can be understood by having a look at data.js
6. The server is supposed to send data in the same structure, it can query over this data but should not manipulate it.
7. After npm install, you can create the college database on your system using command node ./src/createDatabase.js.
8. We are supposed to expose one endpoint on http://locahost:8080

    - GET http://locahost:8080/findColleges
    - This request accepts the following parameters: name, state, city, minPackage, maxFees, course and exams such that::
        - name=ABC filter if the college name contains ABC
        - state=DEF filter if the state in which the college is located contains DEF
        - city=ABC filter if the city in which the college is located contains ABC
        - minPackage=10.5 filter if the college's average package is greater then or equals to 10.5 lac/annum
        - maxFees=10 filter if the college's total fees is less then or equals to 10 lac
        - course=ABC filter if the college provides ABC course

        - exam=DEF filter if the college accepts admission through DEF exam.

1. The server should be able to handle invalid values for minPackage, maxFees, any value other then positive number is considered invalid here.
2. Whole code should be written in index.js located in src folder
3. Hint: try to use regex in monogDB.