**Create Rest API**

**With array**

**Steps to create REST API using Express.js to perform CRUD**

**Step 1:**Create a folder:

mkdir <folder name> i.e. mkdir crud-RestAPI

**Step 2:**First, init a npm in it:

npm init -y

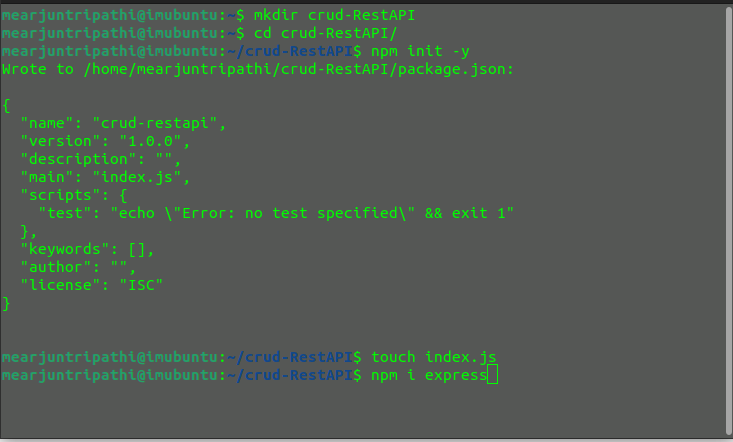
**Step 3:**Create a File in it:

touch <filename.js> i.e. touch index.js (not working)

**Step 4:**Install an express in the package:

npm i express

The updated **package.json** file will look like this:

Process how to setup

**Step 5:**Create a server: Now we **"Building a Student Data Management RESTful API for Data Retrieval, Insertion, Update, and Deletion"**

We are developing a Student Data Management RESTful API that facilitates various operations on student records. With this **API**, you can retrieve a list of all inserted student data, access specific details of individual students, insert new student records, update existing student information, and even delete student records. It's a comprehensive solution for efficiently managing student data within your application.

//import express from 'express'; // for ESM (EcmaScript Module)  
 const express = require('express'); // for CJS (Common JS Modle)  
// as your package type by default it is CJS  
  
const app = express();  
const port = process.env.port || 3000;  
  
let data = [  
 { "name": "Arjun Tripathi", "course": "MCA", "roll\_no": "14", "id": 1},  
 { "name": "Rahul Durgapal", "course": "MCA", "roll\_no": "36", "id": 2 },  
 { "name": "Aman Yadav", "course": "MCA", "roll\_no": "08", "id": 3}  
];  
  
app.use(express.urlencoded({extended:true}));  
app.use(express.json());  
app.listen(port, () => {  
 console.log(`Server is running at: http://localhost:${port}`);  
});

**Explanation:**

1. **Starting a Server and Storing Student Data:**
   * In this code, you are using the Express.js framework to create a web server. The server will listen on a specified port (in this case, it will listen on port 3000 if **process.env.port** is not defined).
   * You have also initialized a data array to store student records. This array acts as an in-memory database where you can store and manipulate student data without using an external database.
2. **app.use(express.urlencoded({extended:true})); and app.use(express.json());:**
   * These lines are configuring middleware in your Express application.
   * **express.urlencoded({ extended: true })** is middleware that parses incoming requests with URL-encoded payloads. It is commonly used to parse data sent by HTML forms.
   * **express.json()** is middleware that parses incoming requests with JSON payloads. It allows your application to handle JSON data sent in the request body.
3. **ESM (EcmaScript Module) and CJS (Common JS Module):**
   * The code you provided uses ESM (import/export syntax) to import the Express.js module. ESM is a module system introduced in newer versions of JavaScript (ES6 and later). It allows you to use **import** and **export** statements to organize and manage your code.
   * The commented line **// const express = require('express');** is an example of using Common JS (CJS) syntax to import the Express module. CJS is the older module system used in Node.js prior to the introduction of ESM.
   * You have a comment that mentions the package type. By default, when you create a Node.js project, it typically uses CJS for module loading. However, modern JavaScript projects often use ESM for its benefits in terms of syntax and static analysis.

In summary, this code sets up an Express.js server, initializes an in-memory data store for student records, and configures middleware for handling URL-encoded and JSON data in incoming requests. It demonstrates the flexibility of Express for handling data without the need for an external database.

Copy paste all endpoints and run node index.js and check in postman

**API endpoints**

**Get All Student Data(Read)**

* **Endpoint:**`/`
* **Method:** GET
* **Description:** Retrieves a list of all student records.
* **Response:** JSON array containing student data.

app.get('/', function (req, res) {  
 res.status(200).json(data);  
});

**Get a Single Student Record(Read)**

* **Endpoint:**\*\* `/:id`
* **Method:** GET
* **Description:** Retrieves a single student record by providing the `id` parameter.
* **Parameters:**
  + `id` (integer): The unique identifier of the student.
* **Response:** JSON object representing the student data if found, or a 404 error if not found.

app.get("/:id", function (req, res) {  
 let found = data.find(function (item) {  
 return item.id === parseInt(req.params.id);  
 });  
 if (found) {  
 res.status(200).json(found);  
 } else {  
 res.sendStatus(404);  
 }  
});

**Insert a New Student Record(Create)**

       { "name": "nishant",

        "course": "course1",

        "roll\_no": "22323"

       }

**Use above as payload and select as json in postman**

* **Endpoint:**`/`
* **Method:** POST
* **Description:** Adds a new student record to the collection.
* **Request Body:** JSON object with the following fields:
  + `name` (string): Student's name.
  + `course` (string): Student's course.
  + `roll\_no` (string): Student's roll number.
* **Response:** Success message with a 201 status code.

app.post('/', function (req, res) {  
  
 let items = data.map(item => item.id);  
  
 let newId = items.length > 0 ? Math.max.apply(Math, items) + 1 : 1;  
  
 let newItem = {  
 id: newId,  
 name: req.body.name,  
 course: req.body.course,  
 roll\_no: req.body.roll\_no  
 }  
  
 data.push(newItem);  
  
 res.status(201).json({  
 'message': "successfully created"  
 });  
});

**Update a Student Record (Update)**

       { "name": "nishant56",

        "course": "course56",

        "roll\_no": "223235"

       }

* **Endpoint:** `/:id`
* **Method:** PUT
* **Description:** Updates an existing student record by providing the `id` parameter.
* **Parameters:**
  + `id` (integer): The unique identifier of the student.
* **Request Body:** JSON object with fields to update:
  + `name` (string): Updated student name.
  + `course` (string): Updated student course.
  + `roll\_no` (string): Updated student roll number.
* **Response:** Success message with a 201 status code if the update is successful, or a 404 error if the student is not found.

app.put('/:id', function (req, res) {  
 let found = data.find(function (item) {  
 return item.id === parseInt(req.params.id);  
 });  
 if (found) {  
 let updateData = {  
 id: found.id,  
 name: req.body.name,  
 course: req.body.course,  
 roll\_no: req.body.roll\_no  
 };  
  
 let targetIndex = data.indexOf(found);  
  
 data.splice(targetIndex, 1, updateData);  
  
 res.status(201).json({ 'message': "data updated" });  
 } else {  
 res.status(404).json({  
 'message': 'unable to insert data because data inserted not matched'  
 });  
 }  
});

**Partially Update a Student Record (Update)**

       {

        "course": "course56912"

       }

* **Endpoint:** `/:id`
* **Method:** PATCH
* **Description:** Partially updates an existing student record by providing the `id` parameter. You can update one or more fields individually.
* **Parameters:**
  + `id` (integer): The unique identifier of the student.
* **Request Body:** JSON object with fields to update (optional):
  + `name` (string): Updated student name.
  + `course` (string): Updated student course.
  + `roll\_no` (string): Updated student roll number.
* **Response:** Success message with a 201 status code if the update is successful, or a 404 error if the student is not found.

app.patch("/:id", function (req, res) {  
 let found = data.find(function (item) {  
 return item.id === parseInt(req.params.id);  
 });  
 if (found) {  
 if (req.body.name) {  
 found.name = req.body.name;  
 }  
 if (req.body.course) {  
 found.course = req.body.course;  
 }  
 if (req.body.roll\_no) {  
 found.roll\_no = req.body.roll\_no;  
 }  
 res.status(201).json({ "message": "data updated" });  
 } else {  
 res.status(404).json({  
 'message': 'unable to insert data because data inserted not matched'  
 });  
 }  
});

**Delete a Student Record**

* **Endpoint:** `/:id`
* **Method:** DELETE
* **Description:** Deletes an existing student record by providing the `id` parameter.
* **Parameters:**
  + `id` (integer): The unique identifier of the student.
* **Response:** No content (204) if the deletion is successful, or a 404 error if the student is not found.

app.delete('/:id', function (req, res) {  
 let found = data.find(function (item) {  
 return item.id === parseInt(req.params.id);  
 });  
 if (found) {  
 let targetIndex = data.indexOf(found);  
  
 data.splice(targetIndex, 1);  
  
 res.sendStatus(204);  
 } else {  
 res.sendStatus(404);  
 }  
});

* Start the server and run the command in your terminal:

node <filename.js> i.e. (node index.js)

* **Testing the API:**To test the **API** endpoints you've defined in your code, you can use tools like[**Postman**](https://www.geeksforgeeks.org/postman-working-http-request-responses/) or **ThunderClient** (an extension for Visual Studio Code).