Task Management RESTful API – Express.js

Submitted by: *Mahesh*

Date: *June 4, 2025*

Assignment: *Build a Basic RESTful API*

Submitted to: *Pratik Dhere*

Organization: *Osumare Marketing Solutions*

A simple and efficient REST API built using Node.js and Express.js to manage a collection of to-do tasks.

Overview

This project is a basic RESTful API designed to manage tasks such as a To-Do list. The goal was to perform **CRUD operations** without a database — by storing tasks in memory.

❖ Technologies Used:

- **Node.js** – JavaScript runtime for server-side development
- **Express.js** – Web framework to create routes and middleware
- **Postman** – API testing tool
- **VS Code** – Development environment

❖ Core Features:

- Add, update, delete, and fetch tasks
- Input validation and error handling
- Authentication (via headers)
- Bonus: Pagination, Sorting, etc

🔐 *All endpoints use proper status codes and follow RESTful standards.*

🛠️ Core Endpoints Summary

Method	Endpoint	Description	Status Codes
GET	*/tasks*	*Retrieve a list of all tasks*	*200, 404*
GET	*/tasks/:id*	*Retrieve a task by its ID*	*200, 404*
POST	*/tasks*	*Create a new task*	*200, 400, 404*
PATCH	*/tasks/:id*	*Update part of a task (e.g., name only)*	*200, 404*
DELETE	*/tasks/:id*	*Delete a task by ID*	*200, 404*

❖ ***Bonus features:***

- *GET /tasks?page=1&limit=2: Pagination*
- *GET /tasks?sortBy=id&order=asc: Sorting*
- *Header Auth: Auth-Key: it's_a_secret*

Project Structure

ASSIGNMENT/

```
|
|— index.js        # Main application file with all logic and routes
|-- package.json     # Project metadata and dependencies
```

All core logic is written inside index.js. As API is relatively small and meant to demonstrate core CRUD operations, I kept the structure clean and focused in a single file to keep it easy to follow.

Code Walkthrough with Request & Response

➢ Initial Data:

Tasks is a Array of Objects which contains:

- *id: Unique number*
- *TaskName: Short title of the task*
- *TaskDesc: Description of the task*

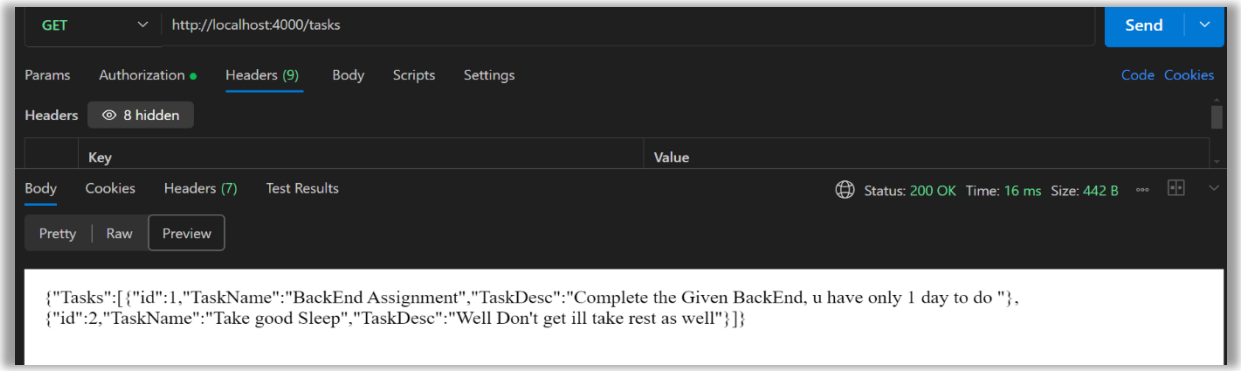
```
// TODO List Array
let Tasks = [
  {
    id: 1,
    TaskName: "BackEnd Assignment",
    TaskDesc: "Complete the Given BackEnd, u have only 1 day to do ",
  },
  {
    id: 2,
    TaskName: "Take good Sleep",
    TaskDesc: "Well Don't get ill take rest as well",
  },
];
```

➢ Endpoint: GET /tasks

- *Fetches data from Tasks array*
- *If no tasks → returns 404 with a message*
- *If tasks exist → returns them with 200 status*

```
// End point to get the TO DO LIST
app.get(
  "/tasks",
  Complexity is 4 Everything is cool!
  /*IsAuth,*/ (req, res) => {
    if (Tasks.length === 0) {
      return res.status(404).json({ msg: "No Tasks are avaliable" });
    }

    return res.status(200).json({
      // "Page No": page,
      // "No Of TAsks Per Page": limit,
      // "Total Pages": Math.ceil(Tasks.length / limit),
      // "To Get next Page Task":
      //   " Update Endpoint like http://localhost:4000/tasks?page=2,limit=2",
      // Task: OnePageTasks,
      Tasks,
    });
  }
);
```

- **req:** Incoming request from client (e.g., new task in req.body)
  res: Server's reply (e.g., success or error using res.json())

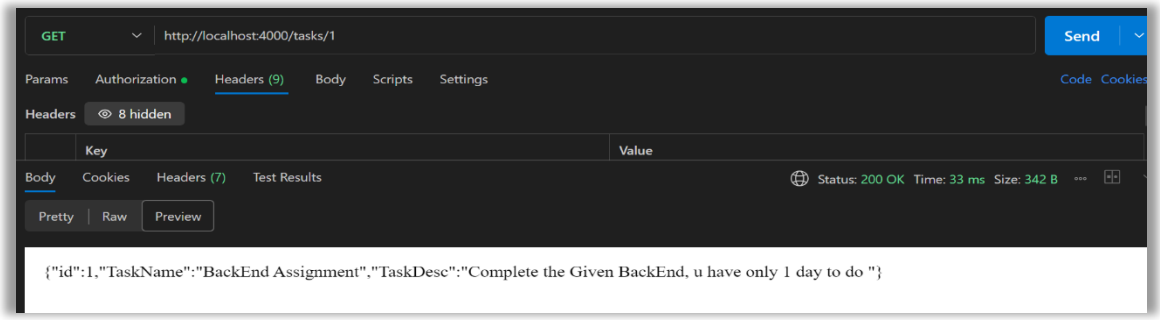


➢ *Endpoint: GET /tasks/:id*

- *Extracts id from the URL*
- *Searches the Tasks array using find()*
- *If task found → returns it with 200 status*
- *If not found → returns 404 with error message*

```
// End Point to get specifc TO DO ITEM by ID
Complexity is 5 Everything is cool!
app.get("/tasks/:id", (req, res) => {
  const { id } = req.params;

  // console.log(id);
  // console.log("Id Type: ", typeof id);

  const task = Tasks.find((t) => t.id === parseInt(id));
  // console.log(task);

  if (!task) {
    return res.status(404).json({ msg: "Task not found or some error" });
  }

  return res.status(200).json(task);
});
```

- ***req:*** *Contains the task ID from the URL → req.params.id*
  res: *Sends back the matching task or an error if not found*

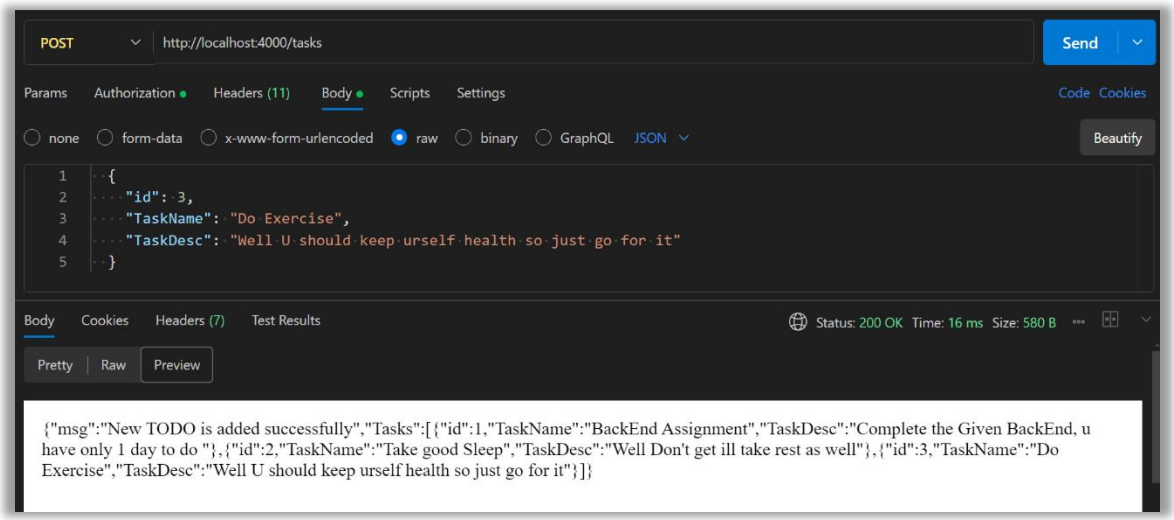


➢ Endpoint: POST /tasks

- *Receives task data from req.body*
- *Validates input and checks for duplicate ID*
- *Adds task to the Tasks array if valid*
- *Returns success response with 200 status*

```
// Post EndPoint to add new TODO Task
Complexity is 11 You must be kidding
app.post("/tasks", IsAuth, (req, res) => {
  // console.log(req.body);
  if (!req.body) {
    return res
      .status(400)
      .json({ msg: "I think u forgot to give the body, give something there" });
  }
  const { id, TaskName, TaskDesc } = req.body;
  // console.log("Id: ", id);
  if (!id || !TaskName || !TaskDesc) {
    return res.status(404).json({ msg: "No Data recieved" });
  }
  const AlreadyExists = Tasks.some((T2) => T2.id === parseInt(id));
  // console.log(AlreadyExists);
  if (AlreadyExists) {
    return res.status(404).json({ msg: "The Task With Same ID alredy exists" });
  }

  // console.log(typeof id);
  Tasks.push({ id, TaskName, TaskDesc });
  return res.status(200).json({ msg: "New TODO is added successfully", Tasks });
});
```

- ***req:*** *Contains new task data → req.body which we sent through Body section in postman*
- ***res:*** *Sends back success message or error*

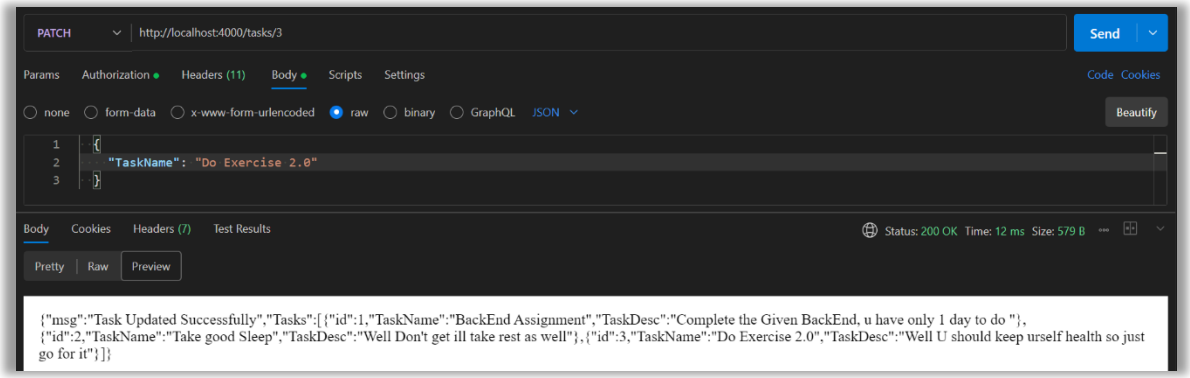


➢ Endpoint: PATCH /tasks/:id

- *Extracts `id` from the URL*
- *Receives updated data in `req.body`*
- *Finds task and updates only provided fields*
- *Returns updated task with 200 status*

```
// Patch Request to update exiting Task
Complexity is 9 It's time to do something...
app.patch("/tasks/:id", IsAuth, (req, res) => {
  let { id } = req.params;
  // console.log(id);
  if (!req.body) {
    return res
      .status(400)
      .json({ msg: "u supoose to send the some data u know" });
  }

  let NTaskName = req.body.TaskName;
  let NTaskDesc = req.body.TaskDesc;
  // console.log(NTaskName, NTaskDesc);

  const TaskExists = Tasks.find((T3) => {
    return T3.id === parseInt(id);
  });
  if (TaskExists) {
    // console.log(NTaskName);
    if (NTaskName !== undefined) TaskExists.TaskName = NTaskName;
    if (NTaskDesc !== undefined) TaskExists.TaskDesc = NTaskDesc;
    // if (NTaskName.length !== 0) TaskExists.TaskName = NTaskName;
    // if (NTaskDesc.length !== 0) TaskExists.TaskDesc = NTaskDesc;
    return res.status(200).json({ msg: "Task Updated Successfully", Tasks });
  }
});
```

- ***req:*** *Has id in req.params, updates in req.body body which we sent through Body section in postman*
- ***res:*** *Sends back success message or 404 if not found*

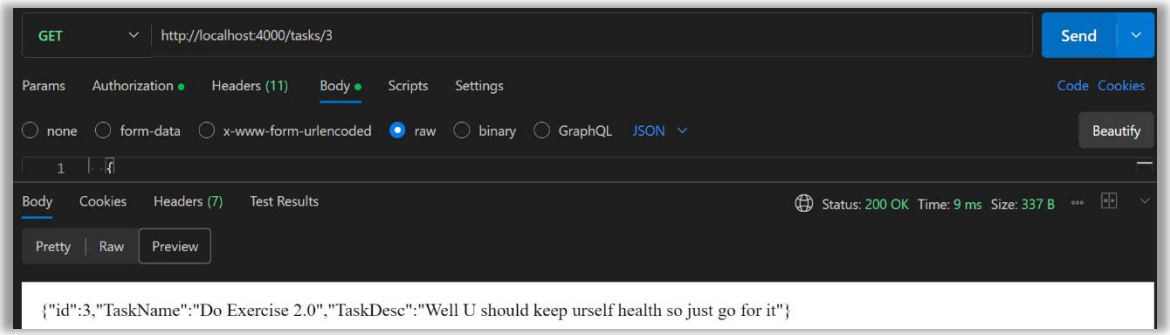


➢ Endpoint: DELETE /tasks/:id

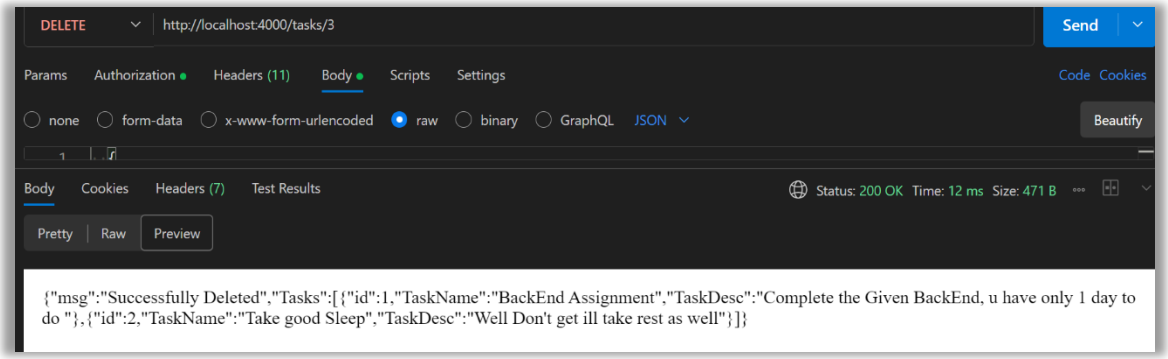
- *Extracts id from the URL*
- *Searches for the task in Tasks array*
- *If found → removes it*
- *Returns success message with 200 status*

```
182  //  Delete Request EndPoint based on the specific ID
     Complexity is 7 It's time to do something...
183  app.delete("/tasks/:id", IsAuth, (req, res) => {
184    const { id } = req.params;
185
186    // console.log(id);
187    const DTask = Tasks.find((T4) => {
188      return T4.id === parseInt(id);
189    });
190    // console.log(DTask);
191    if (!DTask) {
192      return res.status(400).json({ msg: "The Task is already not there" });
193    }
194    Tasks = Tasks.filter((T5) => T5.id !== parseInt(id));
195    return res.status(200).json({ msg: "Successfully Deleted", Tasks });
196  });
```

See the below pic ,we had the Record for id 3



- ***req:*** *Has task ID in req.params.id*
- ***res:*** *Sends confirmation or 404 if not found*



✅ Bonus Feature: Pagination

➢ *Implemented in: GET /tasks endpoint*

- *What is Pagination?*
  Divides the complete list of tasks into smaller pages, reducing load and improving readability.

  - ***How it Works:***
    • Client sends page number and limit → GET /tasks?page=2&limit=2
    • Server calculates which items to return based on those values
    • Returns only the requested portion of the task list

- *We first added more Data Objects there*

```
let Tasks = [
      TaskName: "BackEnd Assignment",
      TaskDesc: "Complete the Given BackEnd, u have only 1 day to do ",
    },
    {
      id: 2,
      TaskName: "Take good Sleep",
      TaskDesc: "Well Don't get ill take rest as well",
    },
    {
      id: 3,
      TaskName: "A1",
      TaskDesc: "A1 Desc",
    },
    {
      id: 4,
      TaskName: "B1",
      TaskDesc: "B1 Desc",
    },
    {
      id: 5,
      TaskName: "C1",
      TaskDesc: "C1 Desc",
    },
  ];
```

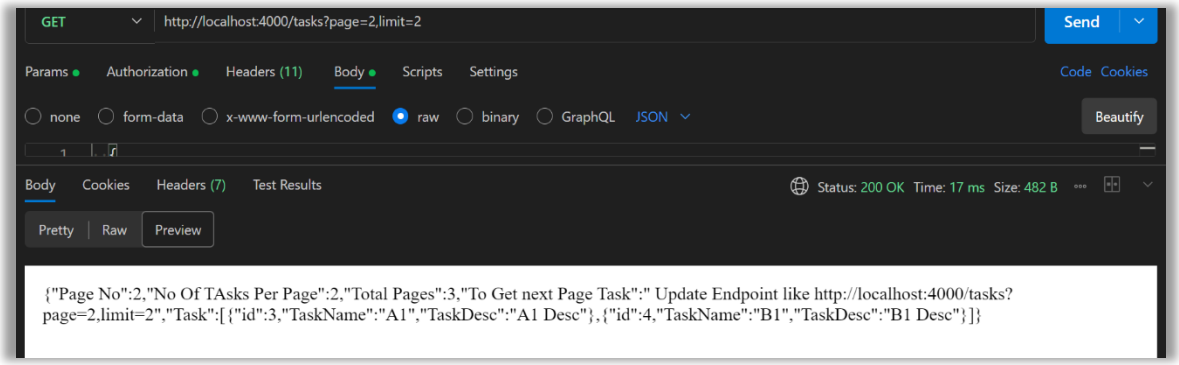
- *Logic:*

```
//
// End point to get the TO DO LIST
app.get(
  "/tasks",
  Complexity is 8 It's time to do something...
  /*IsAuth,*/ (req, res) => {
    if (Tasks.length === 0) {
      return res.status(404).json({ msg: "No Tasks are avaliable" });
    }

    let { page, limit, sortBy, order } = req.query;

    page = parseInt(page) || 1;
    limit = parseInt(limit) || 2;
    // sortBy = sortBy || "TaskName";
```

```
    // Pagination logic
    SIndex = (page - 1) * limit;
    EIndex = page * limit;

    OnePageTasks = Tasks.slice(SIndex, EIndex);

    if (Math.ceil(Tasks.length / limit) < page) {
      return res.status(400).json({
        msg: "That's bad, we don't have that much pages",
        "Pages Avliable": Math.ceil(Tasks.length / limit),
      });
    }
    return res.status(200).json({
      "Page No": page,
      "No Of TAsks Per Page": limit,
      "Total Pages": Math.ceil(Tasks.length / limit)
      "To Get next Page Task":                    Follow link (ctrl + click)
        " Update Endpoint like http://localhost:4000/tasks?page=2,limit=2",
      Task: OnePageTasks,
      // Tasks,
    });
  }
);
```

Can't explain in Detail, In Short

- ***Divides tasks** into pages using page and limit from query.*
- *Calculates start & end index to slice the array*
- *If requested page doesn't exist → returns error.*
- *Else → returns:*
  - *current page,*
  - *tasks on that page,*
  - *total pages,*
  - *info on how to access next page.*



- **Request (req):**

U Notice page and limit in URL, we get response based on it

- **Response (res):**

Returns only the sliced array of tasks → `res.status(200).json(paginatedTasks)`

➢ Bonus Feature: Sorting in GET /tasks

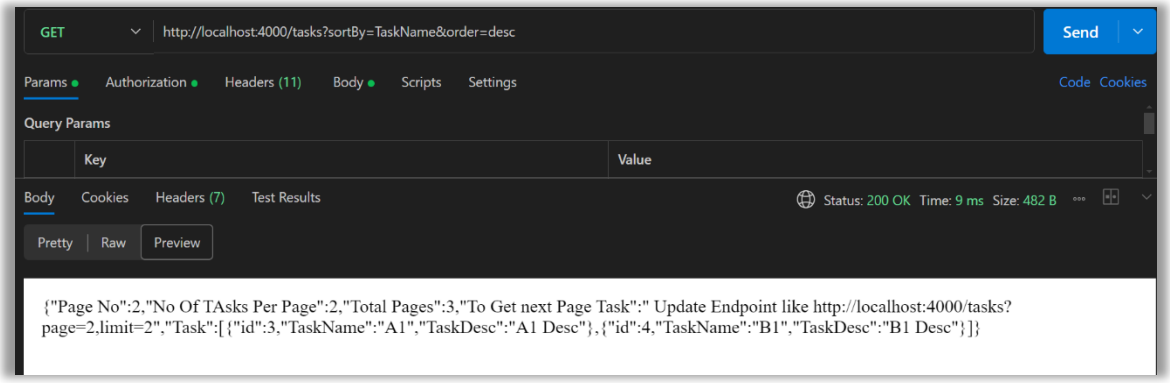
- *Logic*

```
// End point to get the TO DO LIST
app.get(
  "/tasks",
  Complexity is 18 You must be kidding
  /*IsAuth,*/ (req, res) => {
    if (Tasks.length === 0) {
      return res.status(404).json({ msg: "No Tasks are avaliable" });
    }

    let { page, limit, sortBy, order } = req.query;

    page = parseInt(page) || 1;
    limit = parseInt(limit) || 2;
    sortBy = sortBy || "TaskName";
    order = order || "asc";
    // console.log(sortBy, "   ", order);

    //
    //
    // Sorting Logic
    const SortingOpn = ["id", "TaskName", "TaskDesc"];
    if (!SortingOpn.includes(sortBy)) {
      return res.status(400).json({
        msg: "Too bad for u, well we have fields such as (id, TaskName, TaskDesc) only",
      });
    }
    const sortArray = [...Tasks];
    // console.log(sortArray);
```

```
sortArray.sort((a, b) => {
  if (order === "asc") {
    console.log(a[sortBy]);
    return a[sortBy].toString().toLowerCase() >
      b[sortBy].toString().toLowerCase()
      ? 1
      : -1;
  } else {
    return a[sortBy].toString().toLowerCase() <
      b[sortBy].toString().toLowerCase()
      ? 1
      : -1;
  }
});

//
//
// Pagination logic
SIndex = (page - 1) * limit;
EIndex = page * limit;

OnePageTasks = sortArray.slice(SIndex, EIndex);

if (Math.ceil(Tasks.length / limit) < page) {
  return res.status(400).json({
    msg: "That's bad, we don't have that much pages",
    "Pages Avliable": Math.ceil(Tasks.length / limit),
```

```
      });
    }
    return res.status(200).json({
      "Page No": page,
      "No Of TAsks Per Page": limit,
      "Total Pages": Math.ceil(Tasks.length / limit),
      "To Get next Page Task":
        " Update Endpoint like http://localhost:4000/tasks?page=2,limit=2",
      Task: OnePageTasks,
      // Tasks,
    });
  }
);
```

- *Clones task array (to keep original unchanged).*
- *Sorts tasks based on selected field & order using Array.sort().*
- *Converts values to strings and lower case for safe comparison.*
- *sortBy → field to sort (e.g., id, TaskName, TaskDesc)*
  order → asc (A→Z) or desc (Z→A)



(Check the URL how we sent the Params of sortBy and order)

- ***req:***

Contains the query parameters from the URL (like sortBy and order).

- ***res:***

Sends back the sorted data or an error message to the client.

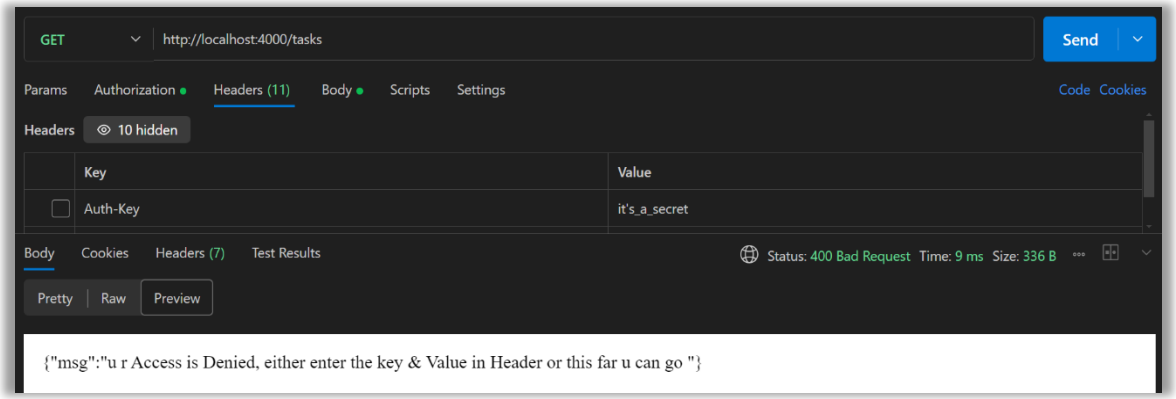
➢ *Bonus Feature: Authorization (Middleware)*

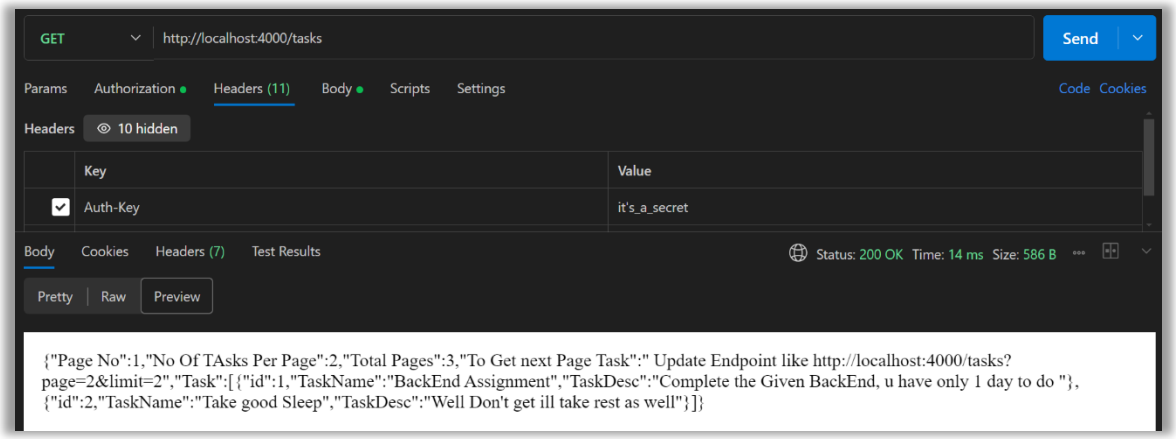
- ***Purpose:*** *Protect sensitive endpoints (e.g., `/tasks`, `/tasks/:id`)*
- ***Header Key Required:*** `Auth-Key: it's_a_secret`
- ***Code:*** *Middleware checks* `req.header("Auth-Key")`
- ***If valid:*** *Calls* `next()` *→ access granted*
- ***If invalid:*** *Responds with* `400` *and access denied message*

- *Logic*

```
//
// Middleware for Authrization on sensetive Endpoints
Complexity is 3 Everything is cool!
const IsAuth = (req, res, next) => {
  const Key = req.header("Auth-Key");
  if (Key !== "it's_a_secret") {
    return res.status(400).json({
      msg: "u r Access is Denied, either enter the key & Value in Header or this far u can go ",
    });
  }
  next();
};

//
//
// End point to get the TO DO LIST
app.get(
  "/tasks",
  Complexity is 18 You must be kidding
  IsAuth, (req, res) => {
    if (Tasks.length === 0) {
      return res.status(404).json({ msg: "No Tasks are avaliable" });
```

- *Below Pics Shows the Res with key & Value in header and without*





Algorithm / Logic Choices (Array Methods Used)

.find()

Used in GET /tasks/:id and PATCH to locate a specific task by matching its id. Helps us fetch or update exactly one task.It directly returns the Single record which we can send or update.

.some()

Used in POST /tasks to check for duplicate IDs before adding a new task — ensures data integrity.As it returns the true if element exists in Array, to avoid duplicate entries.

.filter()
Used in DELETE /tasks/:id to ***remove a task*** *by returning a new array that excludes the one with the specified ID.*

.slice()

Used for pagination in GET /tasks to return a specific portion of tasks (based on page & limit) instead of sending all at once.

.sort()

Used in the same GET /tasks to sort tasks alphabetically or numerically by TaskName, TaskDesc, or id in ascending/descending order.

These methods replicate basic DB operations (find, insert-check, delete, limit, orderBy) without an actual database.

How to Run the Project Locally

1. *Clone the Repository*
   git clone https://github.com/Maheshs-Github/assignment.git
   cd assignment

2. *Install Dependencies*
   npm i

3. *Start the Server*
   node index.js

4. *Base URL*
   http://localhost:4000

5. *Test the API*
   *Use **Postman** or **curl***
   Make requests to endpoints like:
   - *GET /tasks*
   - *POST /tasks*
   - *PATCH /tasks/:id*
   - *DELETE /tasks/:id*
6. *For protected endpoints, **add header**:*
   Key: Auth-Key
   Value: it's_a_secret

✅ Conclusion

- Built a fully functional RESTful API using Node.js & Express
- Implemented **CRUD** operations on in-memory task data
- Enhanced with **Bonus Features**:

  ➤ Pagination

  ➤ Sorting

  ➤ Basic Authorization

- Applied clean code practices using methods like .find(), .some(), .filter() for task operations
- Tested endpoints using **Postman**
- Ensured proper **validation**, **status codes**, and **error handling**

📌 *While no database was used, the design is easily extendable to real-world backend systems.*

Thank You…