

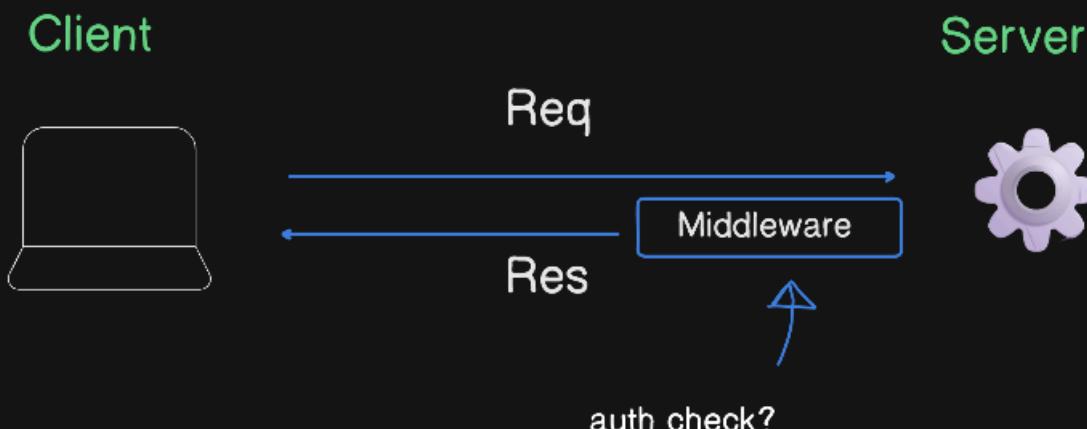
```
export async function getAllNotes(req, res) {
```

Alternative is

```
export async function getAllNotes(_, res) {
```

MIDDLEWARE 💀

Middleware is a function that runs in the **middle** between the **request** and the **response**.



We can do something with the response.

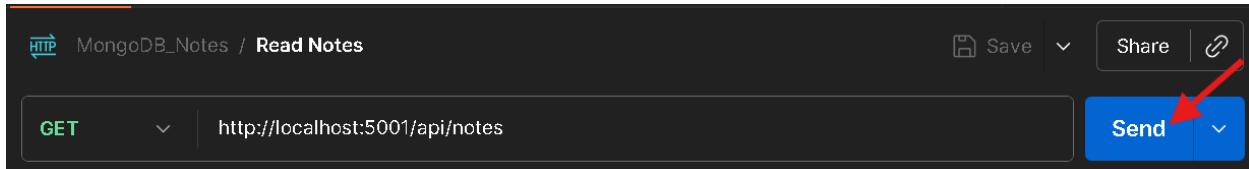
In **server.js**:

```
// middleware
app.use(express.json())

app.use((req, res, next) => {
    console.log("We just got the new req")
    next()
})

app.use("/api/notes", notesRoutes)
```

If in postman we want to get all the notes:



At first we **console log** it["We just got new request"], then **next()** function to call the **getAllNotes** function from get request as given below.

And in [notesRoutes.js](#) has:

A screenshot of a code editor window showing several files: db.js, notesRoutes.js (which is the active tab), server.js, and .env. The notesRoutes.js file contains the following code:

```
router.get("/", getAllNotes)
router.get("/:id", getNoteById)
```

The line `router.get("/", getAllNotes)` has a red underline underneath it.

In [server.js](#):

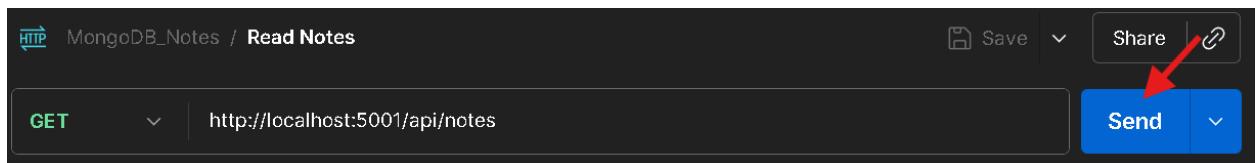
```
n X JS db.js JS notesRoutes.js JS server.js X JS .env JS notesController.js
> JS server.js > app.use() callback

// middleware
app.use(express.json())

app.use((req, res, next) => {
  console.log(`Req method: ${req.method}. \nReq URL: ${req.url}.`)
  next()
})

app.use("/api/notes", notesRoutes)
```

In postman: to get all notes

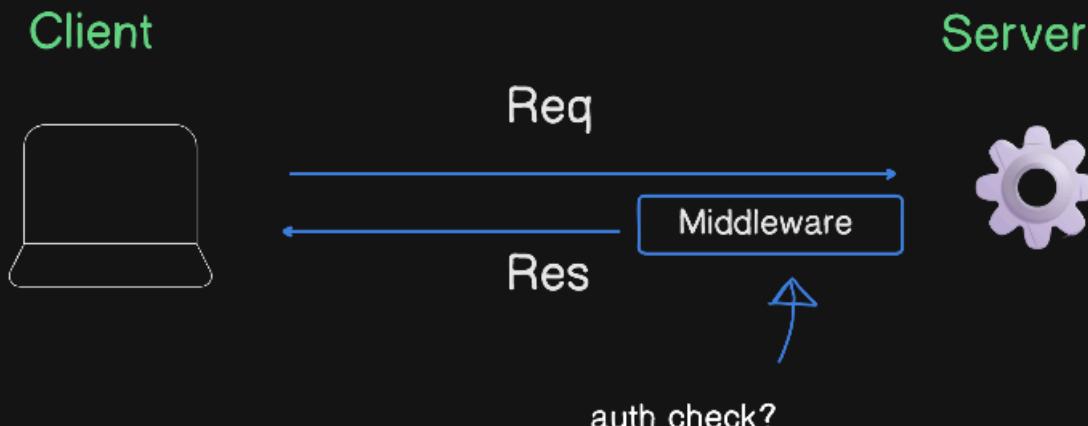


In terminal:

```
[nodemon] restarting due to changes...
[nodemon] starting `node src/server.js`
Server started on PORT: 5001
MongoDB connected Successfully...
Req method: GET.
Req URL: /api/notes.
[]
```

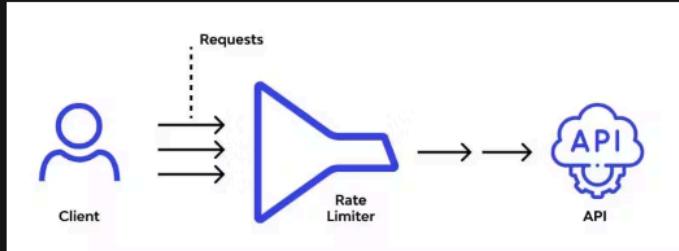
MIDDLEWARE 💀

Middleware is a function that runs in the middle between the **request** and the **response**.



One of the most powerful use case of middle ware is authentication check.

RATE LIMITING



👉 Rate limiting is a way to control how often someone can do something on a website or app like how many times they can refresh a page, make a request to an API, or try to log in.

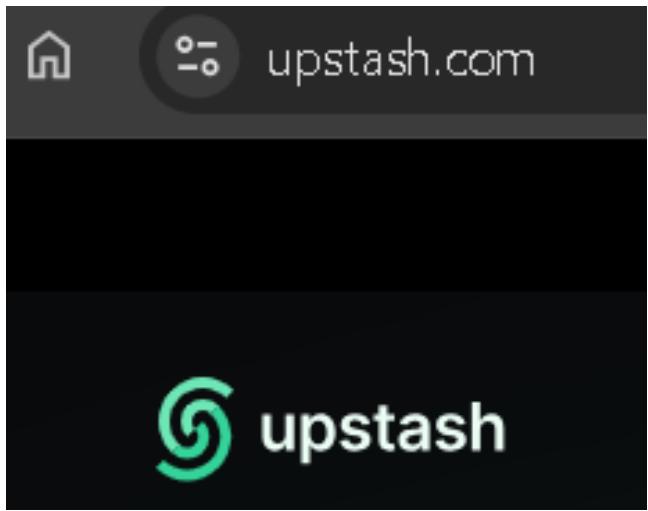
👉 Only 100 requests per user every 15 minutes

RATE LIMITING HELPS WITH

- 👉 Preventing abuse (e.g., stopping someone from making 1000 login attempts in a minute)
- 👉 Protecting servers from getting overwhelmed

429 Too Many Requests

To implement this **rate limiting**, we can use **upstash**:



Visit:

Has a free plan:

Free

\$0



Perfect for prototypes
and hobby projects.

Data Size

256 MB

Monthly Commands

500 K

Start Now

The screenshot shows the upstash Redis homepage. At the top, there are tabs for Redis, Vector, QStash, Workflow, and Search. Below the tabs, the text "Low-latency, serverless key-value store" is displayed. There are two buttons: "Documentation" and "Create Database +". The main content area is divided into three sections: "Highly Available, Infinitely Scalable" (with points ① 99.99% uptime guarantee, ② Automatic scaling to meet your demands, ③ No server management required), "Global Low Latency" (with points ① Lightning-fast response times worldwide, ② Multi-region replication options, ③ Optimize for your users, wherever they are), and "Durable, Persistent Storage" (with points ① In-memory speed with disk-like persistence, ② Data safety without sacrificing performance, ③ Automatic backups).

Login to upstash:

The screenshot shows the login process for upstash. It starts with a "Login" button highlighted by a red arrow. Below it is a "Continue with Google" button with a red arrow pointing to it. Finally, the user's profile information is shown: a profile picture of Routh Kiran Babu, the name "Routh Kiran Babu", and the email "routhfamily123@gmail.com", also highlighted by a red arrow.

Google will allow upstash.com to access this info about you

 Routh Kiran Babu

Name and profile picture

 routhfamily123@gmail.com

Email address

Review upstash.com's [Privacy Policy](#) and [Terms of Service](#) to understand how upstash.com will process and protect your data.

To make changes at any time, go to your [Google Account](#).

Learn more about [Sign in with Google](#).

 Continue

Cancel



Personal

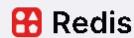
Redis

QStash

Workflow

Vector

...



Redis - Low-latency, serverless key-value store

COMMANDS

0

AVERAGE STORAGE

0 B

COST

\$0.00

 Search...



Import...

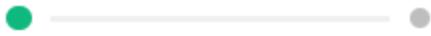
+ Create Database

The screenshot shows a user interface for managing cloud storage or databases. At the top, there's a navigation bar with a green circular icon, a "Personal" dropdown, and several tabs: "Redis" (which is selected), "QStash", "Workflow", "Vector", and "... 4". To the right of the tabs are icons for help, notifications, and a user profile.

The main content area is titled "Redis - Low-latency, serverless key-value store". It displays three summary boxes:

- COMMANDS: 0
- AVERAGE STORAGE: 0 B
- COST: \$0.00

Below these boxes is a search bar with the placeholder "Search...". To the right of the search bar are three buttons: a refresh icon, an "Import..." button, and a prominent green "Create Database" button with a plus sign. A red arrow points to the "Create Database" button.



Create Database

Select a Plan

Name

thinkboard

Primary Region

Mumbai, India (ap-south-1)



Choose the region where most of your writes will take place.

Read Regions

Select read regions (optional)



Read regions are only available for paid plans



Eviction

Enable to evict entries when max data size is reached.



Add a payment method for paid plans.

Cancel

Next



Create Database

Select a Plan

Free

Free forever for hobbyists.

Max Data Size: **256 MB**

Max Monthly Bandwidth: **10 GB**

Pay as You Go

\$0.2 / 100K commands

Fixed 250 MB - 500 GB

Starting from **\$10**

! Add a payment method for paid plans.

Learn more about the plans ↗

Back

Next



Create Database **Select a Plan**

thinkboard
Mumbai, India (ap-south-1)

DEFAULT FEATURES

Persistence REST API TLS Global

Monthly : \$0

! Add a payment method for paid plans.

Back

Create

To connect to the DataBase.

For env file: copy it

Redis / thinkboard ...

Free Tier AWS Mumbai, India ap-south-1 Global Docs SDK

Connect

Connect to your Redis database from anywhere

REST TCP

Read-Only Token

```
1 UPSTASH_REDIS_REST_URL="https://fit-boar-42819.upstash.io"
2 UPSTASH_REDIS_REST_TOKEN="*****"
```

Paste it in .env file:

```
json db.js notesRoutes.js server.js .env notesController.js
.env
1 MONGO_URI=mongodb+srv://routhfamily123_db_user:dRoCgH5MOBbI
2 PORT=5001
3 UPSTASH_REDIS_REST_URL="https://fit-boar-42819.upstash.io"
4 UPSTASH_REDIS_REST_TOKEN="AadDAAIncDI2YTMwOTQwZTA3MTM0ZDk1"
```

Install dependencies:

Stop the server running.

Use the command in backend:

```
npm i @upstash/ratelimit@2.0.5 @upstash/redis@1.34.9
```

```
\backend> npm i @upstash/ratelimit@2.0.5 @upstash/redis@1.34.9
```

In package.json:

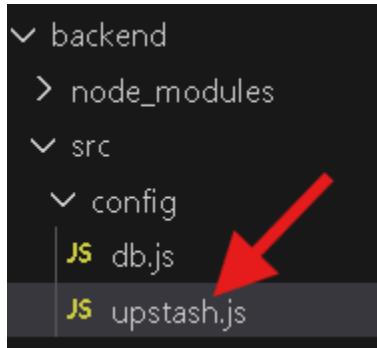
```
{} package.json X js db.js js notesRoutes.js js server.js .env js notesCo
backend > {} package.json > abc main
13   "license": "ISC",
14   "dependencies": {
15     "@upstash/ratelimit": "^2.0.5",
16     "@upstash/redis": "^1.34.9",
17     "dotenv": "^16.5.0",
```

Run the server/app.

```
PS C:\Users\kiran\OneDrive\Desktop\mern-thinkboard\backend> npm run dev
> backend@1.0.0 dev
> nodemon src/server.js

[nodemon] 3.1.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/server.js`
Server started on PORT: 5001
MongoDB connected Successfully...
[]
```

Create upstash under config:



Code in [upstash.js](#):

```
import {RateLimit} from "@upstash/ratelimit"
import {Redis} from "@upstash/redis"

import dotenv from "dotenv"
dotenv.config()

// create a ratelimiter that allows 10 requests per 20 seconds(s)[.slidingWindow(10, "20 s")]
const ratelimit = new RateLimit({
  redis: Redis.fromEnv(),
  limiter: RateLimit.slidingWindow(10, "20 s")
})

export default ratelimit;
```

Best practices:

Create middleware folder:



folder where we can add custom

middlewares.

Code in rateLimiter.js:

```
const rateLimiter = async (req, res, next) => {
```

```
}
```

```
export default rateLimiter;
```

Code in [server.js](#):

```
import rateLimiter from "./middleware/rateLimiter.js"
```

```
connectDB()
```

```
// middleware
```

```
app.use(express.json()) // this mi
```

```
app.use(rateLimiter) 
```

```
^// Our simple custom middleware
```

```
// app.use((req, res, next) => {
```

```
//     console.log(`Req method: ${
```

```
//     next()
```

```
// })
```

```
app.use("/api/notes", notesRoutes)
```

Code in [rateLimiter.js](#):

```
import ratelimit from "../config/upstash.js";
```

```
const rateLimiter = async (req, res, next) => {
```

```
    try {
```

```
        // To check success rate
```

```
        const {success} = await ratelimit.limit("my-limit-key")
```

```
        if(!success){
```

```
            // 429=To many Request
```

```
            return res.status(429).json({message: "To many request, please try again later..."})
```

```

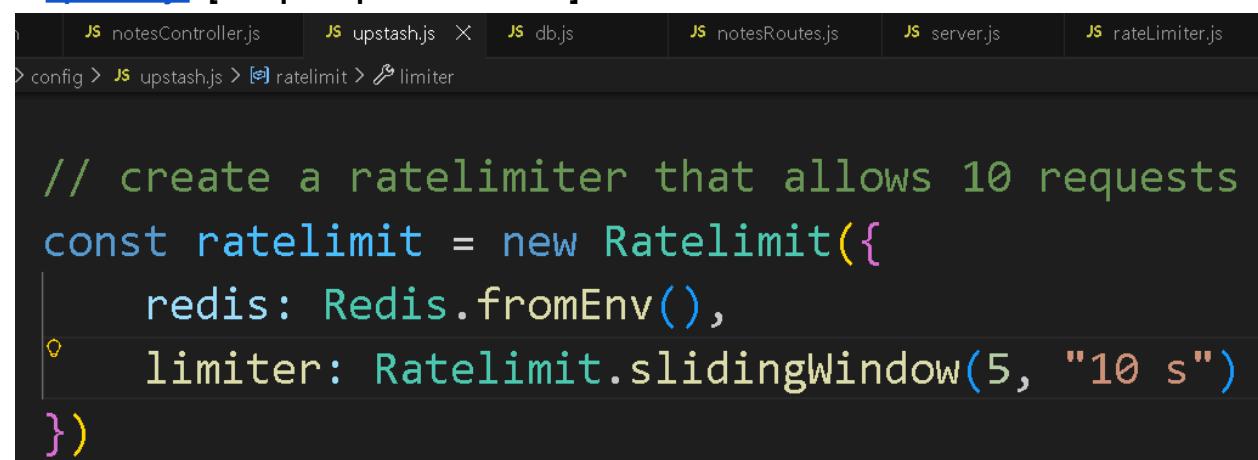
        }
        next() // if success then call the function using next()
    } catch (error) {
        console.log("Rate limit Error", error)
        // we can also add error within next
        next(error)
    }
}

```

export default rateLimiter;

For now let's make the limit as:

In upstash.js: [5 request per 10 seconds]



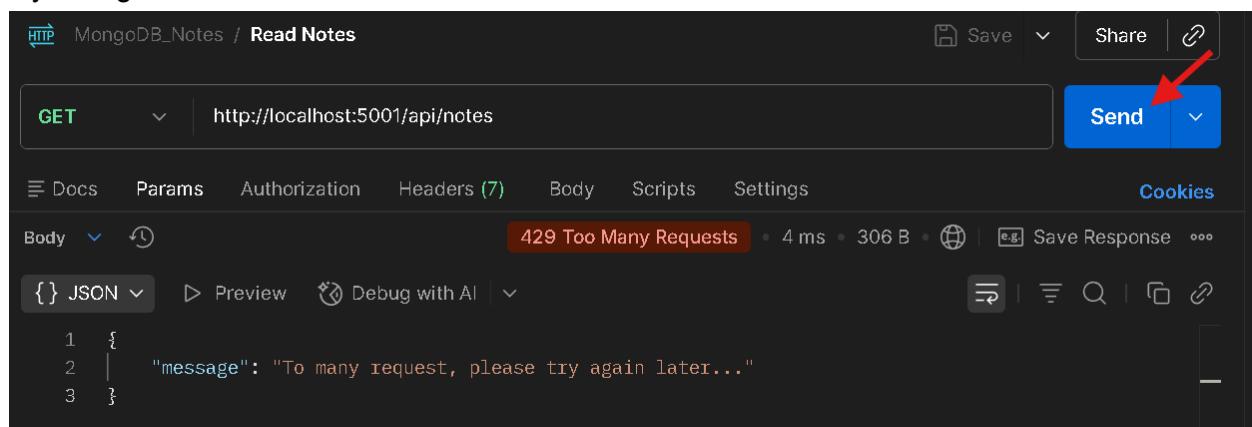
```

// create a ratelimiter that allows 10 requests
const ratelimit = new Ratelimit({
    redis: Redis.fromEnv(),
    limiter: Ratelimit.slidingWindow(5, "10 s")
})

```

In postman:

Try hitting send button more than 5 within 10 Seconds.



Response:

429 Too Many Requests

{ } JSON ▾ Preview Debug with AI

```

1  {
2    "message": "To many request, please try again later..."
3  }

```

As per the code:

```

rateLimiter = async (req, res, next) => {
  // If success
  // 429=To many Request
  return res.status(429).json({message: "To many request, please try again later..."})
}
next() // if success then call the function using next()

```

In terminal:

```

[nodemon] restarting due to changes...
[nodemon] starting `node src/server.js`
Server started on PORT: 5001
MongoDB connected Successfully...

```

At first server is connected then database.

It's better to connect the database, if connected then start the server.

So in server.js , change the code **FROM:**

connectDB()

```

// middleware
app.use(express.json()) // this middleware will parse JSON bodies: req.body
app.use(rateLimiter)
// Our simple custom middleware
// app.use((req, res, next) => {
//   console.log(`Req method: ${req.method}.\\nReq URL: ${req.url}.`)
//   next()
// })

```

app.use("/api/notes", notesRoutes)

```

app.listen(PORT, () => {
  console.log("Server started on PORT:", PORT)
})

```

TO:

```

//connectDB()

// middleware
app.use(express.json()) // this middleware will parse JSON bodies: req.body
app.use(rateLimiter)
// Our simple custom middleware
// app.use((req, res, next) => {
//   console.log(`Req method: ${req.method}.\\nReq URL: ${req.url}.`)
//   next()
// })

app.use("/api/notes", notesRoutes)

connectDB().then(() =>{
  app.listen(PORT, () => {
    console.log("Server started on PORT:", PORT)
  })
})

```

Code in server.js:

```

import express from "express"
import notesRoutes from "./routes/notesRoutes.js"
import { connectDB } from "./config/db.js"

import dotenv from "dotenv"
import rateLimiter from "./middleware/rateLimiter.js"
dotenv.config()

//console.log(process.env.MONGO_URI)

const app = express()
// if process.env.PORT is undefined then PORT = 5001(by default value)
const PORT = process.env.PORT || 5001

//connectDB()

// middleware
app.use(express.json()) // this middleware will parse JSON bodies: req.body
app.use(rateLimiter)
// Our simple custom middleware
// app.use((req, res, next) => {
//   console.log(`Req method: ${req.method}.\\nReq URL: ${req.url}.`)
//   next()
// })

```

```

// })

app.use("/api/notes", notesRoutes)

connectDB().then(() =>{
    app.listen(PORT, () => {
        console.log("Server started on PORT:", PORT)
    })
})

```

Code in upstash.js:

```

import {RateLimit} from "@upstash/ratelimit"
import {Redis} from "@upstash/redis"

import dotenv from "dotenv"
dotenv.config()

// create a ratelimiter that allows 100 requests per minute("60 s")
const ratelimit = new RateLimit({
    redis: Redis.fromEnv(),
    limiter: RateLimit.slidingWindow(10, "20 s")
})

export default ratelimit;

```

Code in rateLimiter.js:

```

import ratelimit from "../config/upstash.js";

const rateLimiter = async (req, res, next) => {
    try {
        // To check success rate
        // "my-rate-limit" <- can provide ip address
        // block based on ip address, not everyone should
        // be blocked if limit exceeds
        const {success} = await ratelimit.limit("my-rate-limit")
        if(!success){
            // 429=To many Request
            return res.status(429).json({message: "To many request, please try again later..."})
        }
        next() // if success then call the function using next()
    } catch (error) {
        console.log("Rate limit Error", error)
    }
}

```

```
// we can also add error within next
next(error)
}
}

export default rateLimiter;
```

Code in .env:

```
MONGO_URI=mongodb+srv://routhfamily123_db_user:dRoCgH5MOBbEmJAW@cluster0.dxejp0q.mongodb.net/notes_db?appName=Cluster0
PORT=5001
UPSTASH_REDIS_REST_URL="https://fit-boar-42819.upstash.io"
UPSTASH_REDIS_REST_TOKEN="AadDAAIncDI2YTMwOTQwZTA3MTM0ZDk1YjgwMWUzYmUwN2RiOGU3YXAyNDI4MTk"
```