How create an API RESTfull in Express (Node.js) without Database

Etienne Rouzeaud · Follow

8 min read · Sep 19, 2018

Listen Share

When you develop a SPA (Single Page Application), the most part of time, you need to contact an API. With a JSON file, Promises, native JavaScript functions and of course, Express, I will show you how to prepare a basic API RESTful.

Prepare the server

Initialized the project

Before, make sure, you have Node.js and NPM installed both.

```
node -v && npm -v
```

Then, create a new project with **npm** command.

```
npm init -y
```

And install packages like **express**, **morgan** and **nodemon**. 📦 📦 📦

```
$ npm install express
$ npm install --save-dev morgan nodemon
```

First route

Create a new file **index.js**.

```
touch index.js
```

And copy paste this code to create a simple server running on port 1337.

```
// Import packages
const express = require('express')
const morgan = require('morgan')

// App
const app = express()

// Morgan
app.use(morgan('tiny'))

// First route
app.get('/', (req, res) => {
    res.json({ message: 'Hello world' })
})

// Starting server
app.listen('1337')
```

Launch server with Nodemon

Open **package.json** in your editor and add a line in the script object, before “test”.

```
"dev": "node_modules/.bin/nodemon -e js",
```

If you use a Node version ≤ 8.2.1, you will need to add the “ — harmony” flag for using spread operator.

```
"dev": "node_modules/.bin/nodemon --harmony -e js",
```

Save modifications and run the command in your terminal.

```
npm run dev
```

Open your navigator and go to http://localhost:1337 or with CURL command.

```
curl -i http://localhost:1337
```

Will returns this result.

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 25
Connection: keep-alive

{"message":"Hello world"}
```

Wonferful! We have an object named “message” with the value “Hello world”.

Our API content

CRUD operations

For each operation, we will need some JavaScript natives functions.

- **Create** : *array.push()*
- **Read** : *array*
- **Read One** : *array.find()* (return an object)
- **Update** : *array.find()*, *array.findIndex()* (return the index)
- **Delete** : *array.find()*, *array.filter()*

For **Create**, **Update** and **Delete**, the data array will be updated in the JSON file.

File organisation

Adding some folders and files like helper, models, routes and data.

```
└── helpers
    ├── helper.js
    └── middlewares.js
└── data
    └──posts.json
└── models
    └── post.model.js
└── routes
    ├── index.routes.js
    └── post.routes.js
```

You can create these folders and files with this command.

```
mkdir helpers && cd helpers && touch helper.js && touch middlewares.js
&& cd ../ && mkdir models && cd models && touch post.model.js && cd
../ && mkdir routes && cd routes && touch index.routes.js && touch
post.routes.js && cd ../ && mkdir data && cd data && touch posts.json
&& cd ../
```

Schema

Let create a blog post (very original...) with these fields in a row.

- **id** : *Number* (unique and increment)
- **created_at** : *Date* (ISO 8601)
- **updated_at** : *Date* (ISO 8601)
- **title** : *String*
- **content** : *String*
- **tags** : *Array*

Data

Example of data in the **posts.json**.

```
[
    {
        "id": 1,
```

```
        "title": "First post",
        "content": "Lorem Ipsum",
        "tags": ["tag1", "tag2", "tag3"],
        "createdAt": "Mon Aug 27 2018 15:16:17 GMT+0200 (CEST)",
        "updatedAt": "Mon Aug 27 2018 15:16:17 GMT+0200 (CEST)"
    },
    {
        "id": 2,
        "title": "Second post",
        "content": "Lorem Ipsum again",
        "tags": ["tag2", "tag4"],
        "createdAt": "Mon Aug 27 2018 16:17:18 GMT+0200 (CEST)",
        "updatedAt": "Mon Aug 27 2018 16:17:18 GMT+0200 (CEST)"
    }
]
```

Helper

Open the **helper.js** file.

```
const fs = require('fs')

const getNewId = (array) => {
    if (array.length > 0) {
        return array[array.length - 1].id + 1
    } else {
        return 1
    }
}

const newDate = () => new Date().toString()

function mustBeInArray(array, id) {
    return new Promise((resolve, reject) => {
        const row = array.find(r => r.id == id)
        if (!row) {
            reject({
                message: 'ID is not good',
                status: 404
            })
        }
        resolve(row)
    })
}

function writeJSONFile(filename, content) {
    fs.writeFileSync(filename, JSON.stringify(content), 'utf8', (err)
=> {
        if (err) {
            console.log(err)
```

```
        }
    })
}

module.exports = {
    getNewId,
    newDate,
    mustBeInArray,
    writeJSONFile
}
```

We have 4 functions very useful for models.

- **getNewId** : searching in the array the last id and increment of 1 to return a new id.
- **newDate** : return the date of your server in ISO 8601.
- **mustBeInArray**: return a promise. Using when we need to check if a row exist via the id (**Read One**, **Update** and **Delete**).
- **writeJSONFile** : write new array in the JSON File data.

Middlewares

Open the **middlewares.js** file.

```
function mustBeInteger(req, res, next) {
    const id = req.params.id

    if (!Number.isInteger(parseInt(id))) {
        res.status(400).json({ message: 'ID must be an integer' })
    } else {
        next()
    }
}

function checkFieldsPost(req, res, next) {
    const { title, content, tags } = req.body

    if (title && content && tags) {
        next()
    } else {
        res.status(400).json({ message: 'fields are not good' })
    }
}
```

```
module.exports = {
    mustBeInteger,
    checkFieldsPost
}
```

We have 2 functions very useful for routes.

- **mustBeInteger** : check before to continue if the id is an integer. Using when we need to get the id (**Read One**, **Update** and **Delete**).
- **checkFieldsPost** : check before to continue if data. Using when we need to get the id (**Create** and **Update**).

Model

Open the **post.model.js** file.

```
const filename = '../data/posts.json'
let posts = require(filename)
const helper = require('../helper.js')

function getPosts() {}
function getPost(id) {}
function insertPost(newPost) {}
function updatePost(id, newPost) {}
function deletePost(id) {}

module.exports = {
    insertPost,
    getPosts,
    getPost,
    updatePost,
    deletePost
}
```

We load the JSON data file and the helper file. Then we prepare 4 functions.

- *getPosts*
- *getPost*
- *insertPost*

- *updatePost*
- *deletePost*

All these functions return a promise. And we don't forget to export them at the end of the file.

getPosts

We have just to return the data array of objects, if array exists.

```
function getPosts() {
    return new Promise((resolve, reject) => {
        if (posts.length === 0) {
            reject({
                message: 'no posts available',
                status: 202
            })
        }

        resolve(posts)
    })
}
```

We return a promise.

- If no posts, display a custom message (***reject,*** **202** ⚠️).
- If posts, diplay array posts (***resolve,*** **200** 👌).

getPost

Like the previous function, except, we want return an object instead an array. For that, we will use the native JavaScript function *find()* for retrieve the object by id in parameter of the function.

```
function getPost(id) {
    return new Promise((resolve, reject) => {
        helper.mustBeInArray(posts, id)
        .then(post => resolve(post))
        .catch(err => reject(err))
```

```
        })
    }
```

We return a promise.

- If not post with this id, display an error message (*reject*, 404 ⚠️).
- If post, display array posts (*resolve*, 200 👌).

insertPost

We will insert a new row.

```
function insertPost(newPost) {
    return new Promise((resolve, reject) => {
        const id = { id: helper.getNewId(posts) }
        const date = {
            createdAt: helper.newDate(),
            updatedAt: helper.newDate()
        }
        newPost = { ...id, ...date, ...newPost }
        posts.push(newPost)
        helper.writeJSONFile(filename, posts)
        resolve(newPost)
    })
}
```

We return a promise.

- If post, display array posts (*resolve*, 200 👌).

updatePost

We will use the native JavaScript function *find()* for retrieve the object by id in parameter of the function. Like adding a post, we have some content from the client. Then we find the index of the row via the native function findIndex. In this row, we add the id, the updated date and the content.

```
function updatePost(id, newPost) {
    return new Promise((resolve, reject) => {
        helper.mustBeInArray(posts, id)
        .then(post => {
```

```
            const index = posts.findIndex(p => p.id == post.id)
            id = { id: post.id }
            const date = {
                createdAt: post.createdAt,
                updatedAt: helper.newDate()
            }
            posts[index] = { ...id, ...date, ...newPost }
            helper.writeJSONFile(filename, posts)
            resolve(posts[index])
        })
        .catch(err => reject(err))
    })
}
```

We return a promise.

- If no post with this id, display an error message (*reject,* 404 ⚠️).
- If post, display array posts (*resolve,* 200 👌).

deletePost

For that, we will use the native JavaScript function *find()* for retreive the object by id in parameter of the function. After checking if the id is an integer and the row exists in the array, we delete via the native function *filter()*.

```
function deletePost(id) {
    return new Promise((resolve, reject) => {
        helper.mustBeInArray(posts, id)
        .then(() => {
            posts = posts.filter(p => p.id !== id)
            helper.writeJSONFile(filename, posts)
            resolve()
        })
        .catch(err => reject(err))
    })
}
```

We return a promise.

- If not post with this id, display an error message (*reject,* 404 ⚠️)
- If post, diplay array posts (*resolve,* 200 👌)

Routes

Open the **index.routes.js** file for declaring the route "/api/v1/posts".

```
const express = require('express')
const router = express.Router()
module.exports = router

router.use('/api/v1/posts', require('./post.routes'))
```

Then in the **index.js** root file for loading the previous file.

```
const app = express()
app.use(morgan('tiny'))
app.use(express.json())
app.use(express.urlencoded({ extended: true }))
app.use(require('./routes/index.routes'))
```

And now, we can work in the **post.routes.js** file.

```
const express = require('express')
const router = express.Router()
const post = require('../models/post.model')
const m = require('../helpers/middlewares')
// Routes
module.exports = router
```

All posts

```
/* All posts */
router.get('/', async (req, res) => {
    await post.getPosts()
    .then(posts => res.json(posts))
    .catch(err => {
        if (err.status) {
            res.status(err.status).json({ message: err.message })
        } else {
            res.status(500).json({ message: err.message })
        }
```

```
    })
})
```

CURL request (200)

```
curl -i -X GET http://localhost:1337/api/v1/posts
```

A post by id

```
/* A post by id */
router.get('/:id', m.mustBeInteger, async (req, res) => {
    const id = req.params.id

    await post.getPost(id)
    .then(post => res.json(post))
    .catch(err => {
        if (err.status) {
            res.status(err.status).json({ message: err.message })
        } else {
            res.status(500).json({ message: err.message })
        }
    })
})
```

CURL request (200, 404, 400).

```
curl -i -X GET http://localhost:1337/api/v1/posts/1
curl -i -X GET http://localhost:1337/api/v1/posts/404
curl -i -X GET http://localhost:1337/api/v1/posts/string
```

Insert a new post

```
/* Insert a new post */
router.post('/', m.checkFieldsPost, async (req, res) => {
    await post.insertPost(req.body)
    .then(post => res.status(201).json({
        message: `The post #${post.id} has been created`,
        content: post
    }))
```

```
        .catch(err => res.status(500).json({ message: err.message }))
    })
```

CURL requests (201, 400).

```
curl -i -X POST \
    -H "Content-Type: application/json" \
    -d '{ "title": "test again", "content": "Lorem Ipsum", "tags":
["tag1", "tag4"] }' \
    http://localhost:1337/api/v1/posts

curl -i -X POST \
  http://localhost:1337/api/v1/posts
```

Update a post

```
/* Update a post */
router.put('/:id', m.mustBeInteger, m.checkFieldsPost, async (req,
res) => {
    const id = req.params.id

    await post.updatePost(id, req.body)
    .then(post => res.json({
        message: `The post #${id} has been updated`,
        content: post
    }))
    .catch(err => {
        if (err.status) {
            res.status(err.status).json({ message: err.message })
        }
        res.status(500).json({ message: err.message })
    })
})
```

CURL requests (200, 404 , 400, 400).

```
curl -i -X PUT \
    -H "Content-Type: application/json" \
    -d '{ "title": "The first", "content": "Lorem Ipsum 2", "tags":
["tag1", "tag4"] }' \
    http://localhost:1337/api/v1/posts/1
```

```
curl -i -X PUT \
    -H "Content-Type: application/json" \
    -d '{ "title": "The first", "content": "Lorem Ipsum 2", "tags":
["tag1", "tag4"] }' \
    http://localhost:1337/api/v1/posts/404

curl -i -X PUT  http://localhost:1337/api/v1/posts/1
curl -i -X PUT  http://localhost:1337/api/v1/posts/string
```

Delete a post

```
/* Delete a post */
router.delete('/:id', m.mustBeInteger, async (req, res) => {
    const id = req.params.id

    await post.deletePost(id)
    .then(post => res.json({
        message: `The post #${id} has been deleted`
    }))
    .catch(err => {
        if (err.status) {
            res.status(err.status).json({ message: err.message })
        }
        res.status(500).json({ message: err.message })
    })
})
```

CURL requests (200, 404, 400).

```
curl -i -X DELETE http://localhost:1337/api/v1/posts/1
curl -i -X DELETE http://localhost:1337/api/v1/posts/404
curl -i -X DELETE http://localhost:1337/api/v1/posts/string
```

Conclusion

After made fun with Promises, build his own API RESTfull without Database still limited. In our case, if we want adding some authors. How to make the relationship ? This is more simple with a real database. You can switch to Database easily with modify the model file.

For a better API, you can use the Joi package to check your model schema and using this middleware for enabling the CORS.

And yes, this API is available on Github.

JavaScript Express API Json Promises

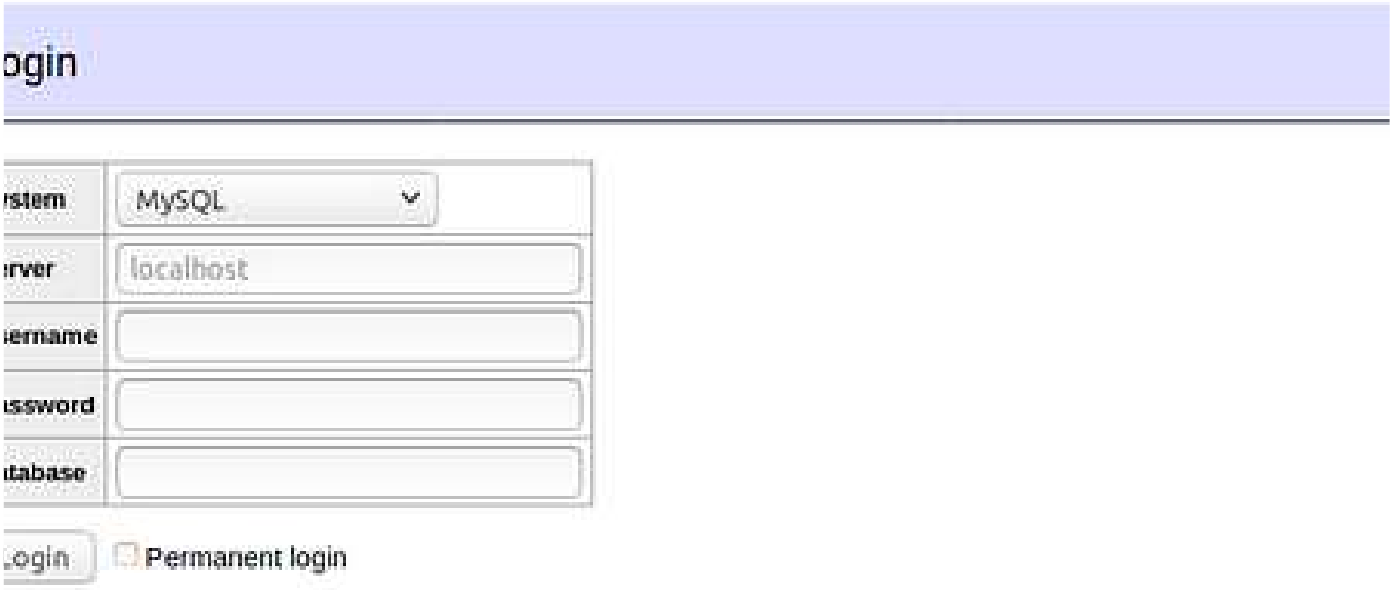
Follow

Written by Etienne Rouzeaud

247 Followers

A curious guy

More from Etienne Rouzeaud



Etienne Rouzeaud

Play databases with Adminer and Docker

When you use Docker with a database you need a client to operate on it. A simple and clean solution is to install a client in a Docker...

Etienne Rouzeaud

A simple CRUD application with Javascript

Let's make a basic application without JS framework to display a simple list of countries. Then we will should make classics actions like...

3 min read · May 6, 2016

896 10

Etienne Rouzeaud

How to create a basic Restful API in Go

without authentication

5 min read · Mar 12, 2015

336 14

Etienne Rouzeaud

Golang YAML to JSON with Gin

You have a YAML file and you want to display these informations in a HTTP API with a JSON result. So you need to read the file then binding...

2 min read · Nov 20, 2016

 12  1 

See all from Etienne Rouzeaud

Recommended from Medium

Melih Yumak in JavaScript in Plain English

Nodejs Developer Roadmap 2023

Explore nodejs developer roadmap for 2023. A step-by-step guide to how to become nodejs developer, increase knowledge as nodejs developer

✦ · 7 min read · Jan 29

600 12

Reed Barger in Web Dev Hero

7 React Projects for Beginners in 2023 (+ Code)

You're ready to start making simple projects with React, but you don't know what to make. Where should you start?

✨ · 6 min read · Jan 11

167 2

Lists

Stories to Help You Grow as a Software Developer

19 stories · 22 saves

Company Offsite Reading List

8 stories · 3 saves

Staff Picks

302 stories · 61 saves

Razvan L in Dev Genius

Build a JSON Body-Parsing Middleware in Node.js

In Express, a middleware is a special type of function that allows to intercept incoming HTTP requests before they reach the controller. It

✦ · 5 min read · Dec 7, 2022

75

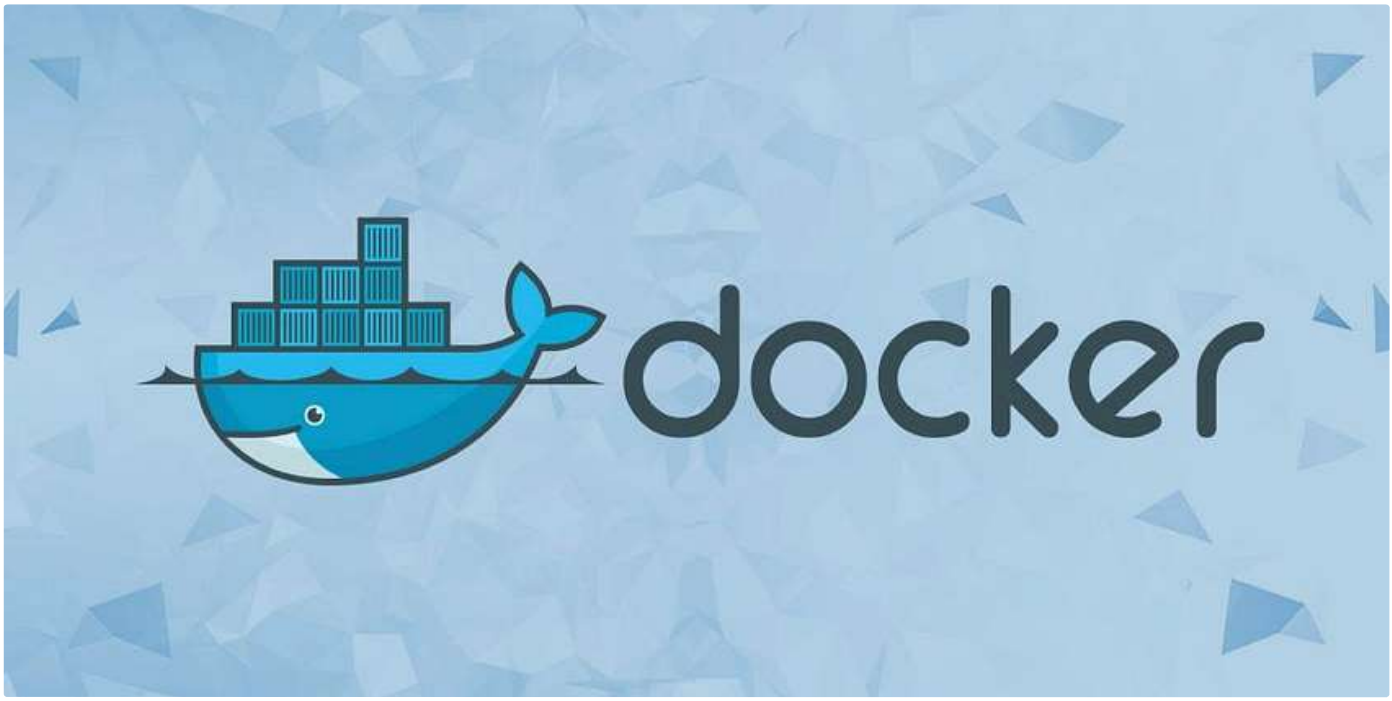
Ibrahim Ahmed in Bootcamp

How I Optimized An API Endpoint To Make It 10x Faster

When it comes to building web applications, performance is one of the most important factors to consider. If your application is slow...

✦ · 3 min read · Jan 11

276 7



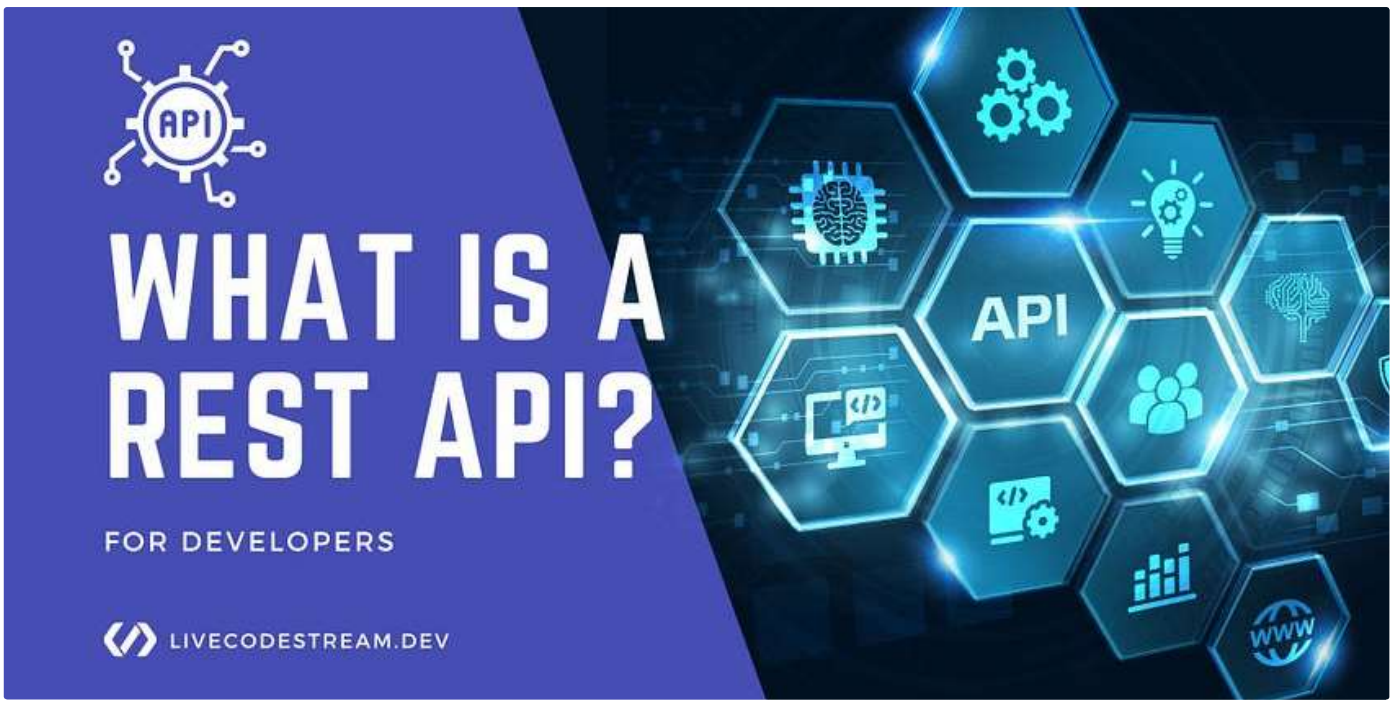
Razvan L in Dev Genius

Run a Secure SSH Server With Docker in 3 Steps

Learn how to run a secure SSH server within a Docker container in 3 easy steps.

· 4 min read · Jan 25

152



Juan Cruz Martinez in Geek Culture

What is a REST API?

What is a REST API? What is RESTful? How do I build an API? Find out these and more in this comprehensive guide on REST APIs.

✨ · 12 min read · Jan 19

 2 

See more recommendations