APIs and Express Routers

Date
Page

Lets list all the api which will we need.

- Post /signup
- Post /login
- Post /logout
- Get /Profile/view
- Patch /Profile/edit
- Patch /profile/password

- Post /request/send/interested/:userId
- Post/request/send/ignored/:userId
- Post/request/review/accepted/:requestId
- Post/request/review/rejected/:requestId

Get /connection
Get /request/received
Get /feed - Get you the profile of the other user on platform.

Right now how we are managing APIs in very very bad ways.

Write now we doing we have just one app.js file and writing everything over here.

Suppose in future we have 40 api's shall we write all api's in one file. Big No
This is very Bad and poor way of doing it.

What we do is we create Express router. and we handle routing in a proper way.

Date
Page

Express Router.

we will use expressrouter to manage our api's ~~effiently~~ efficiently and we will group our api's in different types of routes.

we have 13 API's we will group this api's into small small categories and we will create a seprate router for them and router will handle that route.

authRouter
- Post /signup
- post / login
- Post / logout

Best industry practice is to group the APIs and create different router.

good way

Profile Router
- Get /Profile/view
- Patch / Profile / edit
- Patch / Profile / password.

You can group any api how ever you want to.
(try to do logical seprations)

ConnectionRequest
- Post / request / send / intrested / :userid
- post / request / send / ignored / :userid
- post / request / review / accepted / :requestid
- Post / request / review / rejected / :requestid.

It doesn't matter which api belongs to which router

~~User~~ API - User Router
Get /user / connections
Get / request / received
Get / feed.

usr

API should be very clear. just by reading this we should know what our API is doing.

To manage routing we will do following.
and create a routes folder.

routes ← it will manage routs of different API

→ Auth.js
↳ routes specific to auth.
login / signup / logout

```
const express = require ('express');
const authRouter = express.Router()
```

↑ To create router.
eg express.Router()

```
authRouter.get('/')
app.get('/')
```
} Both almost work same as the end user. it have different logic behind the scene.

```
authRouter.post("/signup", async
                (req, res) => {...});
```

Add all Auth router into this.

```
module.exports = authRouter;
```

```
~~const~~ app = express();
const router = express.Router();
```
} almost same.

Concepts are same.
Everything works the same way.

there will be almost no difference.

Date
Page

create profile router.

```
const express = require ("express")
const ProfileRouter = express.Router();

ProfileRouter.get ("/Profile"), (req, res) => {....});

module.exports = ProfileRouter.
```

now we have seted all the API to different files and routers, but we havent linked it yet.

we have to input router. in app.js.

in app.js

```
const authRouter = require(" auth file")
const ProfileRouter = require.(" auth. ")
const request.js = require ("filepath")
```

now we will use this routes.

now we can use this Routes as middleware
check for all if have the route else check next

```
app.use ("/", authRouter)
app.use ("/", ProfileRoute)
app.use ("/", requestRoute)
```

when ever the request is comming to '/',
go to auth router, and check is there is any route
if yes then use this else skip to next one

I have created 3 apis.

auth.js

Errors.

Error message Exposure –

Const token = ~~async~~ await . userData.getJwt();

↑ even we are using Schema method we have to use await

~~res. res.send(~~

res.cookie("token", token);

↑ we give parameter and key so we can extract token from cookie.

and in catch block we dont send ~~Error~~ throw new Error. we will send error to user.

Profile

Jwt decoding

to decode Jwt we use

Jwt.verify( )

Const decodtoken = await jwt.decode(token, "PassKey")

use ~~verify~~

use (verify) ().

If reading cookies always use (cookieParser) to parse to make it readable format.

To find data by id or anything you have to use or write where to find.

← import it.

const userData = await User.findById(_id)

↑ write or give schema name in our case it's User.

request.js

Some error.

use cookie parser.

find in schema.

- user.findById(_id)

use create / log out api.

outside.

profileRouter.patch("/profile/edit", async (req, res) =>
=> {
try {

const user = req.user

Object.keys(req.body).forEach(key => loggedInUser[key] = req.body[key])

---

for Patch.

(Authorization and send user data via req.user)

↓

Sanitization and validation of data — so validation step.

↓

Loop all the data and override to save the data.

Object.keys(req.body).forEach(key => (loggedInUser[key] = user.body[key]));

↓

Save it to database

Object.keys(req.user).forEach((key) => loggedinuser[key] = req.user[key]);

const isValidToEdit = Object.keys(req.body).every((key) => approvedToEdit.includes(key))

To change password

~~Validate~~

Auth user → check entered password → we will create hash pass of new password → and update it to db.

① userAuth
↓
Data Sanitization
↓
Data Validation.
↓
verified current password
↓
Checked New Password === & Confirm password.
↓
encrypted into hash(newPassword)
↓
Set loggedinuser = hashPassword.