11] Routes & Controllers

Page No.
Date

⇒ Here, all Theory & production gread setup done mostly.

- Now, we have create the code to using that all of early discussed assets.

⇒ In, Controllers folder we write first controller.

→ file in controllers → user.Controller.JS

→ Import the asyncHandler which we made for error Handling.

import {asyncHandler} from "../util/asy...

→ make example of register user. to see how we write routes & api.

```
const registerUser = asyncHandler (
    async (req, res) => {
        res.status(200).json({
            message: "ok"
        })
    })
export {registerUser}
```

Page No.
Date

⇒ Routes : Now we, have write routes for that controllers.

→ routes → user.routes.js file.

- import {Router} from "express";
- import { registerUser} from "controllers/user.co"
  const router = Router()

router.route("/register").post(registerUser)

router.route("/login").post(loginUser)

so, on

export default router

⇒ after made route and controller where we declare it ?

→ so, the best way to declare is in app.js file where all middle ware defined and it also define by middle ware like as...

→ Im, app.js

{{{ }}}

all other early config middlewares (JSON, static, urlencoded) etc

after yet

→ // route import

import userRouter from "./routes/user.route";

// routes declaration

app.use("/api/v1/users", userRouter)

→ standard practice to write api to define version.

→ the declaration define that they pass to user router when "api/v1/users" in the url so it pass to user.route.js where further api are /register, /login, /forgot etc,

→ whole url makeup like as

http://localhost:8000/api/v1/users/register

12] Logic Building

⇒ we have to make register controller it is problem ? (How we do it ?)

→ So, we do it by the defining algo ethm (steps)

① get user data from front-end or postman
② Validation - not emkty.
③ check email or username already exist
④ check for images, avtar is there?
⑤ upload them to cloudinary. C.
⑥ create user obj - create entry in db
⑦ remove pass and refToken field from response
⑧ check for user creation
⑨ return res.

⇒ Now, we have to implement all above point in code. in user.controller.js between the below function,
const registerUser = asyncHandler(...

⇒ we just write here the same imp notes we write at whole code.

→ AS, discuss this is the process that How we decide before writing controller.

→ Eg. body i give the data at front end.

⇒ For files we used multer middleware which we need use it use. routes.js file.

```
router.route("/register").post(
    upload.fields([
        {
            name: "avatar",
            maxCount: 1
        },
        {
            name: "coverImage",
            maxCount: 1
        }
    ]),
    registerUser
)
```

The field which we give frontend input name & backend

There are other profile we can use according to our requirement

→ So, this added multer in our register controller as middleware.

② ⇒ For Validation we use our ApiError

```
if (fullName === "") {
    throw new ApiError(400, "FullName is required")
}
```

=> Tips for checking error. conditional.

```
if ( [ fullName, email, userName, password]
  .some ((field) => field?.trim() === "",
  ))
    throw New ....
  }
```

(8) => Checking the user is exist in DB?

```
const existUser = User.findOne({
  or: [{username}, {email}]
})

if (existUser) {
  throw new ApiError (409, "User already")
}
```

(9) => Handeling files / Image

→ when we use multer as middleware it injecting req files object and we can access all details of the image as given below / accomble below

→ like req.body give form data multer add in req.file give our file data.

→ so, we handle the checking and uploading on cloudinary.

⑤ const avatarLocalPath = req.files?.avatar[0]?.path;

const coverImageLocalPath = req.files?.coverImage[0]?.path;

we check because it may not exist field.

if (!avatarLocalPath) {
    throw new ApiError(400, "Avatar file is")
}

const avatar = await uploadOnCloudinary(avatarLocalPath)
const coverImage = await uploadOnCloudinary(coverIm

if (!avatar) {
    throw new ...
}

⑥ const user = await User.create({
    fullName,
    email,
    password,

from cloudinary ext.url

avatar: avatar.url
coverImage: coverImg?.url || ""
username: username.toLowerCase()
});

(7)

const createdUser = await User.
findById(user._id).select("-password
-refreshToken");

→ The above query is fit for checking the user created by it's _id with findById methods. more on it give chaining on Select method we give we Not want "" field so it give user data except "-field".

(8)

return res.status(201).json(

(9)

new ApiResponse(200, createdUser,
"User Registered Successfully"));

⇒ So, we can make as all easier.

13] ~~Post~~ Postman.

- The professionally it used by sending the api collection between front-end backend.

- Set up the environment Variable & Repeating url to make Variables.

- making collection with endpoint folder.

- eg: apa collection → user api folder, product api folder etc.