INCOME-EXPENSES TRACKER

1. Create two folders one for the server side(server) and one for the client side(client).
2. Add package json (npm init)… entry point : server.js (this file contains the scripts that’s going to help us to run our application and also for us to maintain other packages that we are going to use in our project)
3. For API development we need some folders. i.e. config controllers model routes utils server.js.
4. Creating our express server. (npm i [express@4.18.1](mailto:express@4.18.1)) .
   1. const express = require('express')
      1. require('express') loads the Express module into your project, making its functionality available through the express variable.
   2. const app= express();
      1. (instance of express application)The app object returned by express() represents your web application. This object will be used to define routes, configure middleware, and handle requests and responses.
   3. const PORT = process.env.PORT || 9000;

app.listen(PORT, console.log(`Server is up and running on port ${PORT}`));

* + 1. First line sets the PORT variable to the value of the PORT environment variable if it exists; otherwise, it defaults to 9000. The PORT environment variable is a way to tell your server which port to use for incoming connections. Hosting services set this automatically so your app runs smoothly without needing to change the code.
    2. **app.listen(PORT)**: This method is used to start the Express server. The server will listen for incoming requests on the port number defined by the PORT variable, which could be set by the environment or default to 9000 . The second argument(console.log) of app.listen is a callback function. This function is executed after the server successfully starts listening on the given port.
  1. modify the package.json file : The responbsibility of our **server.js** is to start the server

"main": "server.js",

  "scripts": {

    "start": "node server.js"

1. Backend Dummy routes: for making dummy routes we have to determine how many routes are required based on the project.For this project we require 3 ie userRoute, createAccRoute, and transaction route
   1. For users route to register user we neet post request so

//POST/api/v1/users/register

app.post('/api/v1/users/register', async(req,res)=>{

    try {

        res.json({msg:'Register Route'})

    } catch (error) {

        res.json(error);

    }

});

* + 1. App.post: This method is used to define a route that specifically handles HTTP POST requests. POST requests are typically used when sending data to the server, such as when submitting a form or uploading a file. **Syntax:** app.post(path, callback). **callback**: This is the function that will be executed when a POST request is made to the specified path. A callback function is a function that is passed as an argument to another function and is executed (or "called back") at a later time, usually after some operation is completed.
    2. **URL Path:** This string specifies the endpoint at which the server listens for POST requests. In this case, it’s /api/v1/users/register
    3. async (req, res) => { ... }: The async keyword is used to define an asynchronous function. It allows you to write code that can handle tasks that take time to complete (like fetching data from a database or calling an external API) without blocking the rest of your code. Inside an async function, you can use the await keyword to wait for these tasks to finish before moving on to the next line of code.
    4. **req (Request Object):** Represents the incoming request to the server. It contains data sent by the client (e.g., form data, headers, query parameters).  **Usage:** You can access request data through properties like req.body (for POST data), req.params (for URL parameters), and req.query (for query strings).
    5.  **res (Response Object):** Represents the response that will be sent back to the client. **Usage:** You can use res to send data back to the client using methods like res.send(), res.json(), or res.status().
    6. How it works: When a POST request is made to /api/v1/users/register, Express routes it to the function provided. In the try block, the server responds with a JSON object saying "Register Route". This is the expected response for a successful request. If something goes wrong during the execution (e.g., a database error), the catch block will execute. It sends the error details back to the client as a JSON response, which can help with debugging or understanding what went wrong.
  1. Post method for /users/login
  2. app.post('/api/v1/users/login', async(req,res)=>{
  3. try {
  4. res.json({msg:'Login Route'})
  5. } catch (error) {
  6. res.json(error);
  7. }
  8. });

1. Get method to see the user profile from the server
2. //GET/api/v1/users/profile/id
3. app.get('/api/v1/users/profile/:id', async(req,res)=>{
4. try {
5. res.json({msg:'Profile Route'})
6. } catch (error) {
7. res.json(error);
8. }
9. });
   * 1. HTTP GET requests are usually used to **retrieve data** from the server.
     2. The :id part is a **route parameter**. It means that whatever value is in this part of the URL will be accessible in your code as req.params.id. For example, if the URL is /api/v1/users/profile/123, then req.params.id will be '123'.
10. Delete method to delete the user id
11. //Delete/api/v1/users/profile/id
12. app.delete('/api/v1/users/:id', async(req,res)=>{
13. try {
14. res.json({msg:'Delete Route'})
15. } catch (error) {
16. res.json(error);
17. }
18. });
    * 1. When you use a DELETE method with a route like /api/v1/users/:id, you're telling the server to remove the user with the specified ID from the database.
19. Put method to update the user id

app.put('/api/v1/users/profile/:id', async(req,res)=>{

    try {

        res.json({msg:'Update Route'})

    } catch (error) {

        res.json(error);

    }

});

1. The **ACCOUNT ROUTES** we are going to create is for keeping track of income and expenses

//POST/api/v1/accounts

app.post("/api/v1/accounts",async(req,res)=>{

    try {

        res.json({msg:"Create Account Route"});

    } catch (error) {

        res.json(error);

    }

});

//GET/api/v1/accounts/

app.get("/api/v1/accounts",async(req,res)=>{

    try {

        res.json({msg:"Get all Account Route"});

    } catch (error) {

        res.json(error);

    }

});

//DELETE/api/v1/accounts/:id

app.delete("/api/v1/accounts/:id",async(req,res)=>{

    try {

        res.json({msg:"Delete Account Route"});

    } catch (error) {

        res.json(error);

    }

});

//UPDATE/api/v1/accounts/:id

app.put("/api/v1/accounts/:id",async(req,res)=>{

    try {

        res.json({msg:"Update Account Route"});

    } catch (error) {

        res.json(error);

    }

});

For get method we get all acoounts so we don’t need the id.

1. Create the same http methods for **transactions** route

//POST/api/v1/transactions

app.post("/api/v1/transactions",async(req,res)=>{

    try {

        res.json({msg:"Create Transactions Route"});

    } catch (error) {

        res.json(error);

    }

});

//GET/api/v1/transactions/:id

app.get("/api/v1/transactions/:id",async(req,res)=>{

    try {

        res.json({msg:"Get Single Transactions Route"});

    } catch (error) {

        res.json(error);

    }

});

//DELETE/api/v1/transactions/:id

app.delete("/api/v1/transactions/:id",async(req,res)=>{

    try {

        res.json({msg:"Delete Transactions Route"});

    } catch (error) {

        res.json(error);

    }

});

//UPDATE/api/v1/transactions/:id

app.put("/api/v1/transactions/:id",async(req,res)=>{

    try {

        res.json({msg:"Update Transactions Route"});

    } catch (error) {

        res.json(error);

    }

});

1. Install the nodemon inside server directory i.e. (npm i nodemon -D) and manage the package.json to use the nodemon. Nodemon is a development tool that automatically restarts a Node.js application whenever it detects changes in the source files, streamlining the development process. package.json:

 "main": "server.js",

  "scripts": {

    "start": "node server.js",

    "server": "nodemon server.js"

  },

Use : npm run server

1. Testing our endpoints using **thunder client:** So for this we are going to create collections for specific endpoints i.e. create new collection in thunder client of name users, transactions and accounts.
   1. For users route test create 4 requests i.e. register get delete and update and give the url . as an example: [**http://localhost/9000/api/v1/users/register**](http://localhost/9000/api/v1/users/register)(make sure your server is up and running ).200:ok 404:not found
2. Backend MVC Design(Express Routing): Now we have to refactor the code inside our server file to use MVC design pattern files. Firstly we are going to work with the routing using express and finish up the design pattern

Inside the routes folders we are going to create number of folders based on the routes i.e. users, accounts, transactions.inside the routes/users create a JS file named usersRoute.js. Then we need to use the express routing for that in ruserRoute.js :

const express = require('express');

now we need the routing method from express . we can give any names to initialize it but for convenience use usersRoute:

const usersRoute = express.Router();

Imagine you're building a large web application with many features, like user authentication, product management, and order processing. If you put all the route handlers (i.e., the functions that decide what happens when a user visits a specific URL) in one place, the code could get messy and hard to manage. Instead, you can use express.Router() to separate these routes into different modules (files or sections of your code). Each module can handle a specific set of routes, making your code more organized and easier to maintain.Soo..

**express.Router()** is a method in Express that allows you to create a modular, mini Express application to handle specific routes and middleware, helping you organize and manage your application's code more effectively.

Now we have to bring our business logic(callback function) which is in the server.js into the userRoute.js . the userRoute() we have all http method i.e. userRoute.get, userRoute.post . Lets take the business logic into the userRoute.js like this:

//POST/api/v1/users/register

userRoute.post("/register",async(req,res)=>{

    try {

        res.json({msg:'Register Route'})

    } catch (error) {

        res.json(error);

    }

})

Since the business logic of the server js transferred to the userRoute.js we need to export it.

module.exports = userRoute;

The line module.exports = userRoute; is a way to make the userRoute object available for use, in other parts of your application.

Now use the userRoute which is exported in the server.js as a callback function like this in server.js:

app.use('/api/v1/users', usersRoute);

This line makes all routes in usersRoute available under the /api/v1/users path in your app. If a request comes in for /api/v1/users/login, it will look inside usersRoute for a route that matches /login.

**app.use()** is like a way of telling your app, "Whenever a request comes in, check if it matches this path, and if it does, use this middleware or this group of routes to handle it."**Router as Middleware:** When you define routes using express.Router(), you're essentially creating a set of middleware functions grouped together. This router (e.g., usersRoute) can be attached to a specific path using app.use(). Once attached, Express will pass any requests that match that path through this router.

Note: /api/v1/users chahi common endpoint bhayera server.js ma rakhya ani baaki register,login,delete endpoint chahi userRoute.js ma . yo process le chahi pachi v2 banauna sajilo huncha endpoint

Now add the remaining business logic routes i.e..get,put,delete,update into userRoute.js as follows:

//POST/api/v1/users/login

usersRoute.post("/login",async(req,res)=>{

    try {

        res.json({msg:'Login Route'})

    } catch (error) {

        res.json(error);

    }

});

//GET/api/v1/users/profile/id

usersRoute.get("/profile/:id",async(req,res)=>{

    try {

        res.json({msg:'Profile Route'})

    } catch (error) {

        res.json(error);

    }

});

//DELETE/api/v1/users/:id

usersRoute.delete("/:id",async(req,res)=>{

    try {

        res.json({msg:'Delete Profile Route'})

    } catch (error) {

        res.json(error);

    }

});

//PUT/api/v1/users/:id

usersRoute.put("/:id", async(req,res)=>{

    try {

        res.json({msg:'Updated Route'})

    } catch (error) {

        res.json(error);

    }

});