CMPS 350 – Web Development Fundamentals

Lab 08 – Web APIs using Next.js

Objective

- Creating RESTful Web APIs using Next.js
- Next.js 13.2 routing fundamentals and route definition
- Next.js 13.2 route handlers and file conventions
- Testing RESTful APIs using Postman and Chai HTTP

Resources

- Next.js: Getting Started, Routing Fundamentals, Defining Routes, Route Handlers, and File Conventions (`route.js`)
- HTTP response status codes
- Getting Started with Postman

1. Bank API

The goal of this exercise is to develop a RESTful web API using Next.js 13.2 for managing a collection of bank accounts and their associated transactions.

1. Create a Next.js application using `npx create-next-app@latest --experimental-app`. You can use ESLint but do not use TypeScript or the `src/` directory.
2. Open the project directory using Visual Studio Code and test if the application was successfully created by running `npm run dev`.
3. Create a new `data` directory within the `app` directory, and then paste the provided `bank.json` file. This file will be utilized for data manipulation tasks such as adding, updating, deleting, and retrieving data.
4. Within the `api` directory, create the required directories and `route.js` files to define and handle the subsequent API routes:

Method	URL	Description
`GET`	`/api/accounts/?type=type`	returns accounts by `type`; returns all accounts when `type` is not provided
`POST`	`/api/accounts`	adds a new account and returns it
`GET`	`/api/accounts/:id`	returns the account having `id`
`PUT`	`/api/accounts/:id`	updates the account having `id` and returns it
`DELETE`	`/api/accounts/:id`	deletes the account having `id`
`GET`	`/api/accounts/:id/transactions`	returns all transactions for the account having `id`

`POST`	`/api/accounts/:id/transactions`	adds a new transaction for the account having `id` and returns the account

5. Account identifiers are unique and randomly generated by the API using Nano ID.
6. Include a catch-all, `[[...all]]`, route to handle invalid routes.
7. Use a `try...catch` statement to handle server errors for every request.
8. Return a JSON response and status code using `Response.json(body, { status: code })` for every route.
9. Set the correct status code for every response and, when a request fails, include a meaningful message.
10. Test the API using Postman.
11. Create a `bank-api.spec.js` and test the methods of the API using Mocha/Chai and Chai HTTP.

2. Front-end Client

The goal of this exercise is to develop a front-end client that uses the bank API and allows the end-user to manage a collection of accounts and their associated transactions.

1. Develop a client-side (front-end) application to make to manage a collection of bank accounts and associated transactions. The application should have the following four pages:
   1.1. `accounts.html`: A table with all account information and a drop-down list to filter them by type. Accounts with zero balance can be deleted using a button.
   1.2. `new-account.html`: A form to create a new account, specifying the type and initial balance. A message should be displayed when account creation fails, for example, when the balance is not a number.
   1.3. `transactions.html`: A table with all transactions, their type, amount, date, and corresponding account, and a drop-down list to filter them by type.
   1.4. `new-transaction.html`: A form to create a new transaction, specifying the account number, type, and amount. A message should be displayed when a transaction fails, for example, if the balance is insufficient to perform a withdrawal.
2. Include a navigation bar in every page with links to all pages in your application.

Additional Resources

- Introduction to RESTful web services
- What is a REST API?
- Best practices for REST API design