

CMPS 350 – Web Development Fundamentals

Lab 08 – Web API using Next.js

Objective

- Create Web API using Next.js
- Use Next.js routing
- Test Web API 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](#)

Part 1 - Bank Web API

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

1. Create a new folder named **lab08-bank-app** and open it in VS Code.
2. Create a Next.js application using `npx create-next-app@latest --experimental-app`.

To keep it simple, answer No to all questions:

```
✓ Would you like to use TypeScript with this project? ... No / Yes
✓ Would you like to use ESLint with this project? ... No / Yes
✓ Would you like to use `src/` directory with this project? ... No / Yes
✓ What import alias would you like configured? ... @/*
```

3. Run the app using `npm run dev`
4. Create a new **data** directory under **lab08-bank-app**, and then paste the provided **accounts.json** and **trans.js** files. These files will be utilized for reading, adding, updating, and deleting accounts and transactions data.
5. Create **accounts-repo.js** under `/api/accounts` to implement and test the functions to get/add/update/delete accounts in `accounts.json` file and also functions to get/add transactions in `trans.json` file. When adding a transaction make sure to update the account balance.
6. Within the `api` directory, create the required directories and `route.js` files to define and handle the following API routes. The route handlers should use the functions from the **accounts-repo.js**

Method	URL	Description
GET	<code>/api/accounts/</code>	Returns all accounts. If <code>?type=acctType</code> query parameter is passed then use it to filter the accounts to return.
POST	<code>/api/accounts</code>	Adds a new account and returns it
GET	<code>/api/accounts/:id</code>	Returns the account by 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 transaction id.

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

Part 2 – Bank Front-end Client

The goal of this part is to develop a front-end client that uses the bank Web API and allows the end-user to manage a collection of accounts and their associated transactions. The app should have the following four pages:

- accounts.html:** displays a table with all account information and a drop-down list to filter them by type.
Provide a link besides each account to get the account transactions.
Provide a link besides each account to add a new account transaction.
Provide a delete button to allow deleting accounts with zero balance.
- transactions.html:** displays a table of transactions for a particular account including the transaction type, amount, and date.
- new-account.html:** provides a form to create a new account, specifying the type and the initial balance.
- new-transaction.html:** provides 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.