**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

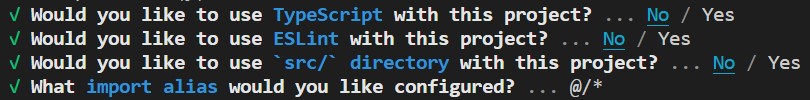
**Resources**

* Next.js: [Getting Started,](https://beta.nextjs.org/docs) [Routing Fundamentals,](https://beta.nextjs.org/docs/routing/fundamentals) [Defining Routes,](https://beta.nextjs.org/docs/routing/defining-routes) [Route Handlers,](https://beta.nextjs.org/docs/routing/route-handlers) and [File Conventions (route.js)](https://beta.nextjs.org/docs/api-reference/file-conventions/route)
* [HTTP response status codes](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)
* [Getting Started with Postman](https://learning.postman.com/docs/getting-started/overview/)

# 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 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:



1. Run the app using **npm run dev**
2. Create a new **data** directory under **lab08-bank-app**, and then paste the provided ***accounts.json*** and ***trans.js*** files. Thess files will be utilized for reading, adding, updating, and deleting accounts and transactions data.
3. 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.
4. 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 | /api/accounts/ | Returns all accounts.  If ?type=acctType query parameter is passed then use it to filter the accounts to return. |
| POST | /api/accounts | Adds a new account and returns it |
| GET | /api/accounts/:accountNo | Returns the account by accountNo |

1 of 2

|  |  |  |
| --- | --- | --- |
| PUT | /api/accounts/: accountNo | Updates the account having accountnp and returns it |
| DELETE | /api/accounts/: accountNo | Deletes the account having accountNo |
| GET | /api/accounts/: accountNo /transactions | Returns all transactions for the account having accountNo |

Adds a new transaction for the

POST /api/accounts/:accountNo/transactions account having accountNo and returns the transaction id.

1. Account identifiers are unique and randomly generated by the API using [Nano ID.](https://zelark.github.io/nano-id-cc/)
2. Include a catch-all, **[[...all]]**, route to handle invalid routes.
3. Use a try...catch statement to handle server errors for every request.
4. 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.

1. Test the API using [Postman.](https://www.postman.com/)
2. Create a bank-api.spec.js and test the methods of the API using Mocha/Chai and [Chai HTTP.](https://www.chaijs.com/plugins/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.