

CMPS 350 Web Development Fundamentals

Lab 9 - Web API

Objective

The objective of this lab is to practice implementing Web API using Node.js and Express. Then test them using Postman and Mocha.

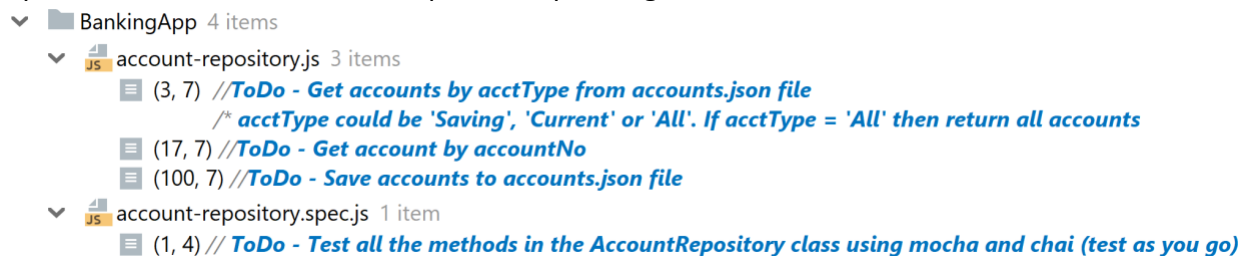
Overview

In this Lab you will develop a Banking App to asynchronously read/write data from the accounts.json file and make the App functionality accessible via Web API.

The Banking App

1. Sync cmps350-lab repo to get the Lab files.
2. Copy **Lab09-WebAPI** folder from cmps350s22-Lab repo to your repository.
3. Open **Lab09- WebApi \BankingApp** in WebStorm. Run **npm install** to install the packages.

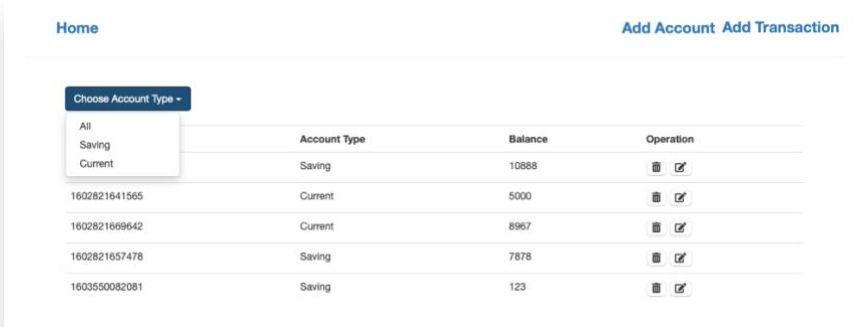
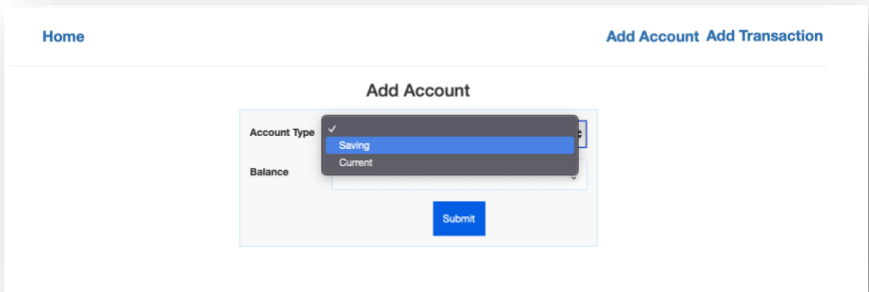
Open the Todo window and complete the pending Todo tasks.



4. Develop Web API to make the App functionality accessible via the web:
 - Run **npm install express** to install express package.
 - Create **app.js** file, import express package, instantiate an express app, configure it then start it at port 3000.
 - Create a **bank-service.js** file and implement handlers for the Urls listed in the table below. The Url handlers should use the methods from the Bank class.

HTTP Verb	Url	Functionality
Get	/api/accounts/?type=acctType	Returns accounts by acctType
Get	/api/accounts/:acctNo	Returns an account by accountNo
Delete	/api/accounts/:acctNo	Deletes an account by accountNo
Post	/api/accounts	Adds an account
Put	/api/accounts	Updates an account
Post	/api/accounts/:id/trans	Add Transaction

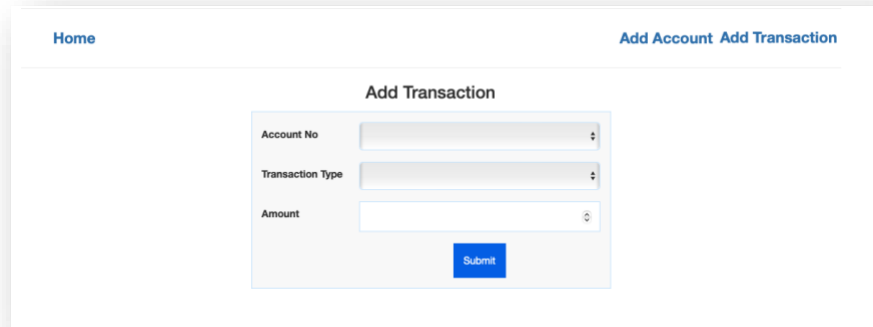
5. Test the Web API using Postman. First download and install it from <https://www.getpostman.com/>
6. Create a **bank-service.spec.js** file. Test the Web API using chai-http. Documentation available at <https://github.com/chaijs/chai-http>
7. Develop a Web UI to make the App functionality accessible to users

Web Page	Functionally
index.html	<p>First, the index page should have a main navigation menu providing 3 links:</p> <ul style="list-style-type: none"> • <u>Home</u> link is the home page that allows getting accounts. • <u>Add Account</u> link allows adding an account. • <u>Add Transaction</u> link allows adding a transaction. <p>- All the app pages should be accessible from the index.html page.</p> <p>- This page allows getting accounts by type. It provides a dropdown to select the account type: <i>Saving</i>, <i>Current</i> or <i>All</i>.</p> <p>When the page load or when a different account type is selected then the page should fetch the accounts from <code>/api/accounts?type=acctType</code></p> <p>- For each account with the balance=0, a <i>Delete</i> button is provided to enable deleting the account. Then this button is clicked the account should be deleted using <code>/api/accounts/:id</code> Web API. Also, the account row should be deleted from the html accounts table.</p> 
acct-form.html	<p>Allows creating a new account by posting the account data to <code>/api/accounts</code></p> 

acct-trans.html

Allows selecting a particular account from the accounts dropdown then submitting a deposit or withdrawal transaction.

The new transaction should be posted to </api/accounts/:id/trans>. The page should display any error returned from the Web API (e.g., insufficient balance).



The screenshot shows a web application interface. At the top, there is a navigation bar with a link labeled 'Home' on the left and two links labeled 'Add Account' and 'Add Transaction' on the right. Below the navigation bar, the main content area is titled 'Add Transaction'. This area contains a form with three input fields: 'Account No' (a dropdown menu), 'Transaction Type' (a dropdown menu), and 'Amount' (a text input field). Below these fields is a blue 'Submit' button.