# Web API exercise

The goal of this exercise is for you to build a simple HTTP Server that passes a set of provide automated scripts. The automated test scripts are provided as [PostMan scripts](https://www.getpostman.com/) and can be run from the command line with [newman](https://www.npmjs.com/package/newman).

## Requirements

The minimum requirement you will have is to be able to build a web server that supports HTTP 1.1.

The solution should be implemented with Node.js. Optionally, a Typescript implementation would be considered a bonus.

We are providing a template that allows you to create the DB and run migrations. Please check the /db/ folder.

To complete the exercise entirely you will need Docker installed. Docker will be used to create and mount the database docker image: <https://www.docker.com/products/docker-desktop>

## How to start the database

We are providing a database with an example migration that creates a dummy table.

You are required to modify the example migration and create the table that's suitable for the exercise specs.

Run the following commands:

* ./db/db-create or, if you're in a NodeJS project npm run db:create - will create the database docker image
* ./db/db-start or, if you're in a NodeJS project npm run db:start - will start the database docker image and run the migrations

You can use a desktop client to connect to the database, depending on your OS, you can use Postico or PGAdmin. Others: <https://linuxhint.com/best_gui_clients_postgresql_ubuntu/>

The default DB connection details are:

* host: localhost
* port: 5432
* username: postgres
* password: postgres
* database: postgres

## Problem space

In this exercise you will build a simple wallet to store and retrieve a *coin* balance for a player.

## Exercise Objectives

You may complete the objectives in any order, but the following is the recommended approach.

### 1. Create a basic web server

Create a web server that responds with a either a 404 or the following balance payload when are request is issued to [http://localhost:8080/wallets/{someid}](http://localhost:8080/wallets/%7Bsomeid%7D)

{

"transactionId": "tx101",

"version" : 1,

"coins" : 100

}



Next implement a Credit endpoint on http://localhost:8080/wallets/{some-id}/credit that takes an HTTP Post request with the following payload and returns a balance.

{

"transactionId" : "tx102",

"coins" : 1000

}

### 2. Implement the domain model

Create an independent module that will host your domain model. This module should have no dependencies. Create a suite of unit tests for this module.

Ideally in a test first approach, create a wallet domain model that will incrementally satisfy the requirements of crediting, debiting, rejecting an excessive Debit, and ensuring an idempotent system.

Part of this work should ideally include setting up your project with a code coverage report.

### 3. Pass all automated test scripts

Now integrate your web server to your domain model. Keeping your domain model free of any dependencies. It should not know about serialization formats, web protocols, or anything but the language it is built with.

The web server now needs to take requests and map them to commands issued to your domain model. Once this is done you should be able to run the web server locally and execute the postman scripts to validate your implementation. You may have to implement an in-memory *repository*.

You can install the Postman GUI from here - <https://www.getpostman.com/>. If you want to run from the command line, then *newman* can be installed via NPM (if you're in a NodeJS project).

npm install -g newman



The scripts can be executed with newman as such:

newman run ./postman/Web-API.postman-collection.json

### 4. Integrate with a data store

If you have created a Repository interface and provided an in memory implementation in the last step, here you can replace it with an implementation that will actually persist to a data store.

The data store provisioning should be scripted.

Use the Postman scripts to validate your implementation.

### Example of successful output

After running the newman cli tool

$ newman run ./postman/Web-API.postman-collection.json



You should expect to see the following output:

newman

Web-API

→ Get Empty Wallet Balance

GET http://localhost:8080/wallets/1d4e7d81-ce9d-457b-b056-0f883baa783d [404 Not Found, 568B, 556ms]

✓ Status code is 404-Not Found

→ Credit Wallet with initial balance

POST http://localhost:8080/wallets/1d4e7d81-ce9d-457b-b056-0f883baa783d/credit [201 Created, 178B, 103ms]

✓ Status code is 201-Created

→ Get Wallet Balance after initial credit

GET http://localhost:8080/wallets/1d4e7d81-ce9d-457b-b056-0f883baa783d [200 OK, 189B, 20ms]

✓ Status code is 200-Ok

✓ Payload as expected

→ Duplicate Credit

POST http://localhost:8080/wallets/1d4e7d81-ce9d-457b-b056-0f883baa783d/credit [202 Accepted, 179B, 23ms]

✓ Status code is 202-Accepted

→ Debit Wallet with more than balance

POST http://localhost:8080/wallets/1d4e7d81-ce9d-457b-b056-0f883baa783d/debit [400 Bad Request, 580B, 23ms]

✓ Status code is 400-Client Error

→ Debit Wallet with less than balance

POST http://localhost:8080/wallets/1d4e7d81-ce9d-457b-b056-0f883baa783d/debit [201 Created, 176B, 29ms]

✓ Status code is 201-Created

→ Duplicate Debit

POST http://localhost:8080/wallets/1d4e7d81-ce9d-457b-b056-0f883baa783d/debit [202 Accepted, 177B, 25ms]

✓ Status code is 202-Accepted

→ Get Wallet Balance

GET http://localhost:8080/wallets/1d4e7d81-ce9d-457b-b056-0f883baa783d [200 OK, 188B, 19ms]

✓ Status code is 200-Ok

✓ Payload as expected

┌─────────────────────────┬──────────┬──────────┐

│ │ executed │ failed │

├─────────────────────────┼──────────┼──────────┤

│ iterations │ 1 │ 0 │

├─────────────────────────┼──────────┼──────────┤

│ requests │ 8 │ 0 │

├─────────────────────────┼──────────┼──────────┤

│ test-scripts │ 8 │ 0 │

├─────────────────────────┼──────────┼──────────┤

│ prerequest-scripts │ 0 │ 0 │

├─────────────────────────┼──────────┼──────────┤

│ assertions │ 10 │ 0 │

├─────────────────────────┴──────────┴──────────┤

│ total run duration: 1004ms │

├───────────────────────────────────────────────┤

│ total data received: 1.06KB (approx) │

├───────────────────────────────────────────────┤

│ average response time: 99ms │

└───────────────────────────────────────────────┘

