NRC CSV Extractor

This Java web service provides an HTTP GET endpoint for downloading the contents of a CSV file named data.csv. It adheres to the HTTP specification and supports various query string parameters and the Accept header.

Features

Provides the ability to download CSV data from a file.

Offers the option to download data in JSON format, with column titles and row data.

Supports specifying the starting row and the number of rows to retrieve.

Installation

To run the application locally, you need to have Java and Maven installed on your system.

Copy the .zip file to your machine.

Navigate to the root directory.

Run the following Maven commands to build and run the application:

mvn clean install

mvn spring-boot:run

By default you can go to [http://localhost:8080](http://localhost:8080/) and view the program.

Usage

Query string parameter support:

The program will retrieve CSV data, send an HTTP GET request to the /download endpoint. You can provide query parameters start (starting row index) and count (number of rows to retrieve).

The current program will perform these GET requests programatically for ease of testing and evaluation. This can easily be done using tools such as Postman.

Accept header support:

The header support comes in the form of retrieving JSON data. Send an HTTP GET request to the /download endpoint with Accept: application/json header. The code can handle cases where even if the start parameter provided is above 0 (Which will skip the column title row in the /download endpoint), it will still provide column titles in the JSON response for readability.

Dependencies

Spring Boot: Framework for building Java-based applications.

OpenCSV: Library for reading and writing CSV files.

JSON: Library for processing JSON data.

Spring Web: Provides web development features.

Implementation considerations:

Are both success and error situations handled in a standard and consistent manner?

Success situations are consistent as the endpoint and “Accept” header and query parameters will result in the correct result. A Json format response and String response written to console respectively.

Errors are caught consistently in the code by leveraging try/catch blocks.

If a much larger CSV file were provided, would the web service still function as designed?

Currently, the application attempts to load the CSV file to memory before parsing. Since the memory is a finite resource, a large enough CSV file will break the program.

To resolve this, it would be ideal to use streaming and “chunks” to parse the CSV file. Or the user must be constrained manually to use smaller CSV files.

Would another developer be able to easily make fixes and changes to the web service?

With the use of Spring and Maven, the project structure is well-understood by other Spring developers as it is standardized. By following clean code principles and utilizing easy-to-understand naming conventions, the code should be easy for another developer to debug and make changes to.

Would a developer or deployer be able to have confidence that the web service is functioning as expected?

The confidence that the web service is functioning as expected comes from my manual testing with the given dataset.

To increase confidence levels, it would be advisable to introduce automated unit testing as further proof of a robust implementation.

Would a deployer be able to easily run the web service?

Again, with the use of Spring boot to implement the given task, user will be able to run the project the “Spring boot way”. Standardized instructions will be easy to follow and can be found in the Readme.