Business Case – Grocery Product Feed Parser

This is my complete solution that fulfills all the requirements and incorporates the bonus points. The project is structured for AWS Lambda compatibility and includes documentation and a testable structure.  
  
This is the structure of the project:  
  
Immagine che contiene testo, schermata, software, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.

The main.py script is designed for an AWS Lambda function that downloads a CSV product feed from a given URL, parses it, and outputs it in a format expected by the Glovo API. The fetch\_csv function retrieves and decodes the CSV data from a URL, converting it into a list of dictionaries. The parse\_products function transforms each row into a structured product object, applying basic validation and availability logic. The handler function orchestrates everything, fetching the URL from the event or environment variable, processing the data, and printing the final result as formatted JSON.

The mock\_server.py script creates a simple mock server that serves a static CSV file at the /products.csv endpoint for testing purposes. It's useful for simulating a real URL when developing and validating the Lambda function locally

The event.json file provides input to the Lambda function, specifying two possible URLs for downloading the CSV: one for local use (url) and one for Docker/SAM local testing (urlSam). It allows the function to dynamically choose the correct source depending on the environment it's running in.

This is how the process should work to test it locally:  
  
1. Run the mock\_server, so that it is up and running, and you can download the CSV file from it with a GET request  
2. Run the main.py script, which will execute a GET in localhost to the mock\_server, downloading the CSV file  
3. Finally, you can see the result displayed in the console:  
Immagine che contiene schermata, software, testo, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.  
  
I can also get the same result even if I use the environment variable, running the script after the definition of the environment variable:  
Immagine che contiene testo, schermata, software, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.

For local sam local invoke testing, you also need a SAM template.yaml file to define your Lambda function. This defines the python version, the handler, the source directory, and an environment variable.   
After installing the AWS SAM CLI and Docker, which SAM uses to emulate the AWS Lambda runtime locally, I had to update the URL, since localhost no longer works from inside the Docker container. By using host.docker.internal, the Lambda function running in Docker can connect to the mock server on my local machine, allowing it to produce the same result:  
Immagine che contiene testo, schermata, software, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.

EXTRA:  
  
To demonstrate and prove that my solution works well, I deployed it on my AWS environment, I ran a test triggering the Lambda with the even.json file containing a link of a public repo in github hosting the CSV file (https://raw.githubusercontent.com/AndreaVentura97/grocery-feed-stack/refs/heads/main/products.csv).  
As you can see in the pictures below, the execution works without errors, it returns the expected result, and in the last picture you can see the logs correctly displayed on CloudWatch:  
Immagine che contiene testo, schermata, software, Icona del computer

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, software, Icona del computer

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, software, schermo

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, software, Icona del computer

Il contenuto generato dall'IA potrebbe non essere corretto.