let’s consider a **data processing pipeline**. The pipeline will process a user-uploaded CSV file, pass the data through multiple filters (like cleaning, transformation, and aggregation), and finally return the processed output to the user.

**Use Case:**

Imagine a scenario where the user uploads a CSV file with raw sales data, and the system processes it by:

1. **Filter 1: Data Cleaning** – Removing invalid entries.
2. **Filter 2: Transformation** – Converting currency values.
3. **Filter 3: Aggregation** – Calculating total sales or averages.

This can be useful in applications such as data analytics tools, sales dashboards, or reporting systems.

**Implementation Steps:**

* A **front-end** where the user uploads a CSV file.
* A **back-end** (Flask) implementing the Pipe-Filter architecture, where each filter performs a specific task on the data.

**Code Breakdown:**

**1. Server Code (Flask with Python)**

Here’s a Python implementation using Flask to handle the Pipe-Filter architecture on the server side:

python

Copy code

from flask import Flask, request, render\_template, redirect, url\_for

import pandas as pd

from io import StringIO

app = Flask(\_\_name\_\_)

# Step 1: Data Cleaning Filter

def clean\_data(data):

# Remove rows with missing or NaN values

return data.dropna()

# Step 2: Transformation Filter (Convert prices from USD to EUR)

def transform\_data(data):

# Assume 1 USD = 0.85 EUR for demonstration

data['Sales (EUR)'] = data['Sales (USD)'] \* 0.85

return data

# Step 3: Aggregation Filter (Calculate total sales)

def aggregate\_data(data):

total\_sales = data['Sales (EUR)'].sum()

return f"Total Sales (EUR): {total\_sales}"

# Pipe-Filter processing function

def process\_file(data):

# Step 1: Clean the data

cleaned\_data = clean\_data(data)

# Step 2: Transform the data

transformed\_data = transform\_data(cleaned\_data)

# Step 3: Aggregate the data

result = aggregate\_data(transformed\_data)

return transformed\_data, result

# Route for the homepage with file upload form

@app.route('/')

def home():

return render\_template('index.html')

# Route to handle the file upload and processing

@app.route('/upload', methods=['POST'])

def upload\_file():

if 'file' not in request.files:

return redirect(request.url)

file = request.files['file']

if file.filename == '':

return redirect(request.url)

if file:

# Read the uploaded CSV file

file\_data = StringIO(file.read().decode('utf-8'))

data = pd.read\_csv(file\_data)

# Process the data using the Pipe-Filter architecture

processed\_data, result = process\_file(data)

# Return the result and processed data to the front-end

return render\_template('result.html', tables=[processed\_data.to\_html(classes='data')], result=result)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**2. Front-End (HTML)**

You’ll need two HTML templates:

1. **File Upload Page (index.html)**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Pipe-Filter Data Processing</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

background-color: #f4f4f4;

}

.container {

text-align: center;

padding: 20px;

background-color: white;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}

input[type="file"] {

margin: 10px 0;

}

input[type="submit"] {

padding: 10px 20px;

background-color: #4CAF50;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #45a049;

}

</style>

</head>

<body>

<div class="container">

<h1>Upload CSV File for Data Processing</h1>

<form action="/upload" method="POST" enctype="multipart/form-data">

<input type="file" name="file" accept=".csv" required>

<br>

<input type="submit" value="Process Data">

</form>

</div>

</body>

</html>

1. **Result Page (result.html)**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Data Processing Result</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

background-color: #f4f4f4;

}

.container {

text-align: center;

padding: 20px;

background-color: white;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}

table {

width: 100%;

margin-bottom: 20px;

border-collapse: collapse;

}

table, th, td {

border: 1px solid #ddd;

padding: 8px;

}

th {

background-color: #f2f2f2;

}

.result {

font-size: 18px;

margin-top: 20px;

color: #333;

}

</style>

</head>

<body>

<div class="container">

<h1>Processed Data</h1>

<div class="table-container">

{{ tables|safe }}

</div>

<div class="result">

<p>{{ result }}</p>

</div>

<a href="/">Go back to upload another file</a>

</div>

</body>

</html>

**How It Works:**

1. **Front-End**: The user uploads a CSV file containing sales data.
2. **Server (Flask)**: The uploaded file is processed through the Pipe-Filter architecture:
   * **Filter 1: Data Cleaning** – Removes rows with missing values.
   * **Filter 2: Transformation** – Converts the sales values from USD to EUR.
   * **Filter 3: Aggregation** – Calculates the total sales in EUR.
3. The processed data and the result (total sales) are sent back to the user on the result page.

**Example CSV File:**

Your CSV might look like this:

mathematica

Copy code

Product,Sales (USD)

Product A,1000

Product B,1500

Product C,2000

Product D,NaN

The pipeline will:

* Remove the row for "Product D" (because of missing data).
* Convert the sales values from USD to EUR.
* Calculate and display the total sales in EUR.

**How to Run the Project:**

1. Save the Python code as server.py.
2. Create a folder named templates in the same directory and place the two HTML files (index.html and result.html) inside it.
3. Install necessary packages:

bash

Copy code

pip install flask pandas

1. Run the Flask server:

bash

Copy code

python server.py

1. Open a browser and go to http://127.0.0.1:5000/.
2. Upload a CSV file and view the processed data.

**Conclusion:**

This demonstrates the **Pipe-Filter Architecture** with a real-life use case of data processing. The pipeline consists of cleaning, transformation, and aggregation filters, which process raw sales data. The front-end allows the user to interact with the system by uploading data, and the Flask back-end processes the data in a structured pipeline.

4o