**Title: File Compressor and Converter**

**Tool Details:**

**Technology Stack:**

* **Frontend:** Web Components (Lit or Vanilla JS)
* **Backend:** Express.js (Node.js)
* **File Processing:** Compress and convert files (e.g., ZIP, TAR, GZ, PDF, DOCX, PNG, JPG)
* **Data Handling:** File uploads & API requests

**Goal:**  
By completing this assignment, candidates will:

* Learn how to build a **RESTful API** using **Express.js**.
* Implement **file compression and format conversion**.
* Work with **Web Components** to create a frontend for file upload and selection.
* Understand **client-server communication** via API calls.
* Gain experience in handling **file processing techniques**.

**Assignment Description:**

Develop a **File Compressor and Converter** where users upload a **file** (e.g., PDF, DOCX, PNG, JPG) via a **WebComponent-based form**. The backend, built with **Express.js**, will allow users to choose an **output format** and apply **compression** before returning the processed file. The frontend should display the **file details** and provide a **download link** for the processed file.

**Tasks & Steps:**

**1. Backend API Development (Express.js):**

* Set up an **Express.js** server to handle **file uploads**.
* Implement **file compression** using a suitable **compression library**.
* Implement **file conversion** for different formats (e.g., DOCX to PDF, PNG to JPG).
* Implement API routes to **process the file and return the converted and/or compressed file**.

**2. Frontend (WebComponent-based UI):**

* Create a **form** using Web Components that allows users to **upload a file**.
* Provide options for **file format conversion** (e.g., DOCX → PDF, PNG → JPG).
* Provide options for **compression levels** (e.g., ZIP, TAR, GZ).
* Send the uploaded file and user-selected options to the backend using the **fetch API**.
* Display the **file details** and provide a **download link** for the processed file.

**3. Integration & Testing:**

* Ensure the frontend **properly sends data** to the backend.
* Handle errors gracefully (e.g., **unsupported file formats, large files**).
* Test **end-to-end functionality** to ensure seamless integration.

**Mathematical Calculation/Steps (if applicable):**

* **Compression Ratio Formula:**  
  CR=Original File SizeCompressed File SizeCR = \frac{\text{Original File Size}}{\text{Compressed File Size}}
* **Conversion Logic:** Depends on file type transformation (e.g., DOCX to PDF conversion).

**Third-Party Packages (if required):**

* express (for backend server)
* multer (for handling file uploads)
* archiver (for compressing files into ZIP/TAR)
* sharp (for image format conversion and compression)
* pdf-lib (for PDF manipulations)
* lit (for WebComponent development)

**Acceptance Criteria:**

* The **Express.js backend** should successfully process **uploaded files**, apply **compression and/or format conversion**, and return the processed file.
* The **WebComponent-based frontend** should have a **responsive form** that allows users to upload files and select **conversion and compression options**.
* Proper **error handling** should be in place for **unsupported file formats**.
* The application should work seamlessly across **modern web browsers**.

**Submission Guidelines:**

1. **Fork** the provided GitHub repository.
2. **Create a folder** named file-compressor-converter-<your-name>.
3. **Implement the backend and frontend** in the respective subfolders.
4. **Push the code** to your forked repository.
5. **Submit a pull request** with a brief description of your implementation.

**Ensure that the backend correctly handles requests, processes the file, and seamlessly integrates with the WebComponent-based frontend.**