Assignment 4

1. Path Module (2 Grades):

Write an HTTP server that handles file paths. Implement the following:

- **Task 1.1:** Respond with a breakdown of a given file path (e.g., extract root, directory, file name, and extension) and return the full path in a formatted string.
  - **URL:** POST /path-info (Get the file path from the body)
  - **Input:** Provide a file path, such as C:/Users/example/project/sample.txt.
  - **Output:**


```
{
    "parsedPath": {
      "root": "C:/",
      "dir": "C:/Users/example/project",
      "base": "sample.txt",
      "ext": ".txt",
      "name": "sample"
    },
    "formattedPath": "C:/Users/example/project/sample.txt"
}
```


- **Task 1.2:** checks if a provided file path is absolute and returns additional path-related information (Ensure that you use path.join() to handle the correct path separators for your file system (i.e., using / or \ as appropriate)).
  - **URL**: POST /path-check
  - **Input:** Provide a relative or absolute file path in the request body (e.g., ./data/file.txt or /Users/example/project/data/file.txt).
  - **Output:**


```
{
    "isAbsolute": false,
    "basename": "file.txt",
    "extname": ".txt",
    "joinedPath": "./folder/data/file.txt",
    "resolvedPath": "/Users/example/project/data/file.txt",
}
```

```
{
    "isAbsolute": true,
    "basename": "file.txt",
    "extname": ".txt",
    "joinedPath": "/Users/example/project/folder/file.txt",
    "resolvedPath": "/Users/example/project/data/file.txt"
}
```

Assignment 4

2. Events Module (2 Grades)

Create an event emitter to handle file operations. Implement the following:

- **Task 2:** Emit an event when a file is created, read, or deleted.
  - **URLs:**
    - POST /create-file (Get the file name from the body)
    - DELETE /delete-file (Get the file name from the body)
  - **Input Example for File Creation:**

```
{
    "fileName": "example.txt",
    "content": "This is a new file."
}
```

  - **Expected Output (Event Logged to the console):**

    "Event emitted: fileCreated for example.txt"

3. OS Module (1 Grades) (Search Point)

Gather system information and return it in the response.

- **Task:** Respond with the system's architecture, platform, free memory, and total memory**.**
  - **URL:** GET /system-info
  - **Expected Output:**

```
{
    "architecture": "x64",
    "platform": "win32",
    "freeMemory": 2147483648,
    "totalMemory": 8589934592
}
```

Assignment 4

4. File System Module (2 Grades)

Perform file-based CRUD operations using a file to store data.

- **Task 3.1:** Create and delete a file.
  - **URLs:**
    - POST /create-file (Get the file name from the body)
    - DELETE /delete-file (Get the file name from the body)
  - **Input Example for File Creation:**

```
{
    "fileName": "notes.txt",
    "content": "Some important notes."
}
```

  - **Expected Output:** The file is created, read, or deleted, and the relevant content or confirmation message is sent back.
  - **Example (After creating the file):**

```
{
    "message": "File created successfully."
}
```

- **Task 3.2**: Asynchronously read from and append to a file. Use path.join() to create the file path and path.resolve() to get the absolute path before performing the read and write operations.
  - **URLs:**
    - POST /append-async (Get the file name from the body)
    - POST /read-async (Get the file name from the body)
  - **Input Example for Asynchronous File Write:**

```
{
    "fileName": "async.txt",
    "content": "This is written asynchronously."
}
```

  - **Expected Output:** The file is written or read asynchronously, with a confirmation message

Assignment 4

5. Streams (3 Grades)

- **Task 5.1** Create a readable stream that reads data from a file (buffer size will be "16" and make it automatically close when it ends) and pipes it to the response. Implement error handling for stream events.
    - **URL:** POST /stream-file (Get the file name from the body)
    - **Expected Output**: Stream the contents of data.txt to the client, logging relevant stream events to the console and if there is any error, return it in the response to the user
    - **Example Console Output**:

      Stream opened

      Data event received: [data chunk]

      Stream ended
- **Task 5.2:** Stream data from one file to another.
    - **Input Example:**

```
{
    "sourceFile": "source.txt",
    "destinationFile": "destination.txt"
}
```

    - **URL:** POST /copy-file
    - **Expected Output:** The file is successfully copied, and a message confirming the operation is sent back.
- **Task 5.3:** Stream Transformation with **Gzip** Compression
    - **Description:** Read a file, compress its content using **Gzip**
    - **URL:** POST /compress-file (Get the file name from the body)
    - **Expected Output:** The compressed file content is stored on your machine and return response with message done to the user**.**

Assignment 4

⚠️ Important Notes about postman

1. **Name the endpoint with a meaningful name like 'Add User', not dummy names.**
2. **Save your changes on each request(ctrl+s).**
3. **Include the Postman collection link (export your Postman collection) in the email with your assignment link**

Bonus (3 Grades)

How to deliver the bonus?

1- Solve the problem Kth Missing Positive Number on **LeetCode**

2- Inside your assignment folder, create a **SEPARATE FILE** and name it “bonus.js”

3- Copy the code that you have submitted on the website inside ”bonus.js” file